VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rekonstrukce 3D tvaru objektu využitím projekce strukturovaného světla

3D Reconstruction Using Structured Light

2017 Bc. David Buček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. David Buček**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Rekonstrukce 3D tvaru objektu využitím projekce strukturovaného světla
3D Reconstruction Using Structured Light

Jazyk vypracování: čeština

Zásady pro vypracování:

3D rekonstrukce scény pomocí strukturovaného světla je moderní přístup užívaný v oblasti počítačového vidění. Máme-li zkalibrovaný systém kamera-projektor, můžeme pomocí promítání speciálních vzorů na snímaný objekt provést jeho 3D rekonstrukci. Projektor osvětluje objekt binárními vzory, přičemž software analyzuje obraz z kamery a na základě hledání korespondujících bodů je schopen triangulovat jejich polohu. Cílem diplomové práce je navrhnout experimentální systém tvořený kamerou a projektorem a implementovat vhodný software pro 3D rekonstrukci scény.

Řešení bude obsahovat:
1. Popis stávajících systémů využívajících strukturovaného světla.
2. Podrobný popis navrženého řešení.
3. Implementaci algoritmu pro rekonstrukci 3D tvaru objektu využitím projekce strukturovaného světla.
4. Testování navržené implementace a vyhodnocení testů.
5. Závěr a zhodnocení výsledků.

Seznam doporučené odborné literatury:

[1] Song Zhang, Handbook of 3D Machine Vision: Optical Metrology and Imaging. CRC Press. 2013. ISBN 1439872198
[2] Richard Szeliski, Computer Vision: Algorithms and Applications, Springer, 2011. ISBN 1848829345
[3] Richard Hartley, Multiple View Geometry in Computer Vision. Cambridge University Press. 2004. ISBN 0521540518

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Ing. Michal Krumnikl, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 21. dubna 2017

.....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 21. dubna 2017

.....

Rád bych na tomto místě poděkoval panu Ing. Petru Bučkovi, za pomoc při sazbě textu a poskytnuté rady při implementaci, dále panu Ing. Josefu Hrabalovi za jeho předané vědomosti v rámci hardwarové části a panu Ing. Mgr. Michalu Krumniklovi, PhD. za jeho příkladné vedení a poskytnuté rady. V neposlední řadě pak také své rodině, přítelkyni a přátelům za podporu během celého průběhu vzniku této práce.

Abstrakt

Základem práce je sumarizovat přehled možností návrhu a využití systému rekonstruujícího 3D tvar objektu za pomocí projekce strukturovaného světla. Cílem je navrhnout a vytvořit experimentální bezkontaktní optický skenovací systém pro automatickou 3D rekonstrukci. Při použití techniky strukturovaného světla je standardem kombinace digitální kamery pro snímání scény a projektoru promítajícího předem definované vzory. Máme-li takovýto systém zkalibrovaný a navržený software analyzuje obraz z kamery, můžeme na základě hledání korespondujících bodů triangulovat jejich polohu v souřadném systému, tedy provést 3D rekonstrukci. Na výstupu navrženého řešení bude mračno bodů (*point cloud*) v některém z běžně používaných formátů.

Klíčová slova: Go, Golang, mračno bodů, rekonstrukce, skener, zpracování obrazu

Abstract

The aim of thesis is to summarize overview of the design and utilization of the system reconstructing 3D shape of an object using structured light projection. The target is to design and build an experimental non-contact optical scanning system for automatic 3D reconstruction. In the technique of structured light, there is a standard using combination of digital cameras to capture scenes and projector to project predefined patterns. If we calibrate the system well, and proposed software analyze the capture from camera, we could on the basis of exploring corresponding points triangulate their position in coordinate system, consequently realize a 3D reconstruction. At the output of the proposed solution, there will be the point-cloud in one of the commonly used formats.

Key Words: Go, Golang, image processing, point-cloud, reconstruction, scanner

Obsah

Seznam použitých zkratek a symbolů

ADAS	–	Advanced Driver Assistance Systems
API	–	Application Programming Interface
ARM	–	Acorn RISC Machine
ASCII	–	American Standard Code for Information Interchange
BOM	–	Bill of Materials
BSD	–	Berkeley Software Distribution
CAD	–	Computer-aided design
CNC	–	Computerized Numerical Control
DLP	–	Digital Light Processing
GNU	–	GNU is Not Unix
GPIO	–	General Purpose Input Output
GPU	–	Graphics Processing Unit
HD	–	High Definition
HDMI	–	High Definition Multimedia Interface
HTML	–	Hyper Text Markup Language
IEEE	–	Institute of Electrical and Electronics Engineers
IP	–	Internet Protocol
JSON	–	JavaScript Object Notation
microSD	–	Micro Secure Digital
MIME	–	Multipurpose Internet Mail Extensions
PNG	–	Portable Network Graphics
PPP	–	Pstros Packed Points
P.S.T.R.O.S	–	Point-cloud System Targeted for Reconstruction of Object Shape
P.S.T.R.O.S.i.C.E	–	Point-cloud System Targeted for Reconstruction of Object Shape In C Extension
RFC	–	Request For Comments
RISC	–	Reduced Instruction Set Computing
SDRAM	–	Synchronous Dynamic Random Access Memory
TFT LCD	–	Thin Film Transistor Liquid Crystal Display
URL	–	Uniform Resource Locator
UTF-8	–	UCS/Unicode Transformation Format
WebGL	–	Web Graphics Library
XGA	–	Extended Graphics Array

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Je tento objekt skutečně čtvercem? Pro lidi, nebo přesněji řečeno pro lidský mozek, je odpověď na tuto otázku již při prvním pohledu poměrně snadným úkolem. Převést tento úsudek do domény strojů a zautomatizovat jej je však mnohem obtížnější. Rozpoznání tvarů vychází z potřeby automatizovaného rozpoznávání objektů, signálů a obrazů nebo z potřeby automatizovaného rozhodování založeného na dané sadě parametrů.

Zpracování obrazu včetně jeho analýzy se jako samostatný vědní obor vyvinul teprve nedávno, konkrétně na začátku 60. let 20. století. Jedná se o obor velmi rozsáhlý, jelikož již ve svém základě zasahuje do mnoha různorodých odvětví, počínaje základními principy statistiky, výpočetní techniky a informatiky, umělé inteligence, a v neposlední řadě matematiky jako takové. Z důvodu, že je oblast tohoto oboru dosti mladá a neprobádaná, anebo proto, že je den ode dne více využívána ve stále dalších odvětvích, jedná se o velmi aktivní oblast výzkumu. Máme-li jmenovat oblasti využití zpracování obrazu, musíme se zaměřit pouze na ty nejvýznamnější, jelikož, jak již bylo zmíněno, se oblast působnosti neustále rozrůstá. Uveďme strojírenství, lékařství či bezpečnostní průmysl (například policie, kriminalistika, bezpečnostní agentury), jelikož právě tam je analýza obrazu využívána nejhojněji.

Vzhledem k neustálému pokroku v informačních technologiích a zvyšování výkonů počítačů, nám současná doba umožňuje vizualizovat virtuální obsah v nevídané kvalitě. Ten dnes máme možnost pozorovat v televizi, animovaných filmech, počítačových hrách, ale i ve stavebním a strojním průmyslu. Přičemž počítačová fikce a zobrazování trojrozměrného světa je již častokrát k nerozeznání od fotografií. A právě pokrok na poli 3D počítačové grafiky zvyšuje poptávku po trojrozměrných počítačových modelech.

Pro vytváření virtuálních modelů je v dnešní době k dispozici množství modelovacích programů, které jsou mnohdy dostupné i se zdrojovými kódy. Zhotovení kvalitního modelu však vyžaduje dostatečné zkušenosti, a také velké množství času. Proto je potřeba mít systémy na rychlou a kvalitní rekonstrukci objektů, které by byly schopné automaticky nasnímat reálný objekt a interpretovat jej jako 3D model. Díky pokroku v přesnosti měřících zařízení, jakož i ve zvyšující se kvalitě ostatních hardwarových součástí, je dnes na trhu dostupných mnoho funkčních systémů pro automatickou 3D rekonstrukci objektů.

Hlavním tématem této diplomové práce je položit a navrhnout základ pro samostatně fungující systém, který by sloužil k rekonstrukci 3D tvaru objektu. Tento systém pak poskytne východisko pro možnou další práci na systému a jeho nasazení v praxi.

První část práce je věnována teoretickým východiskům, které se tohoto tématu týkají.Další kapitola obsahuje rešerši stávajících algoritmů a obdobných systémů, vyskytujících se na trhu. Rozebírá a analyzuje jejich fungování, zajímavá řešení a v neposlední řadě popisuje i jejich výhody a nevýhody. Následující úsek práce již pojednává o vlastním řešení problému. Je zde podrobně popsána architektura systému, její sestavení a fungování. Rovněž i popis procesů probíhajících uvnitř jejích částí a okomentování, proč byl právě takovýto návrh zvolen.

Poslední část je pak věnována testování výsledného řešení. Jsou zde uvedeny testy jednotlivých hardwarových i softwarových částí systémů. Nedílnou součástí je i zhodnocení dosažených výsledků a funkčnosti systému.

2 Teoretické základy

Tato kapitola seznamuje čtenáře se základními pojmy týkající se dané problematiky. Jsou zde zadefinovány termíny mračen bodů, včetně jejich získání, které je soustředěno na strukturované světlo. Ačkoli je práce založena na principech a algoritmech oboru zpracování obrazu a počítačové grafiky, jejich definice není pro tuto práci stěžejní, a proto je zde vynechána. Veškeré dále používané pojmy jsou však dohledatelné v [1, 2, 3].

2.1 Mračno bodů (*point-cloud*)

Tento pojem vychází z oblasti fotogrammetrie a 3D skenování, včetně laserového skenování (LiDAR). Jedná se o velmi rozsáhlou množinu bodů (řádově miliony až miliardy) nesoucí určitou informaci v daném souřadnicovém systému. Doplňující informaci k těmto bodům může být směr normály anebo informace o barvě daného bodu. Stěžejní vlastností všech těchto bodů je, že nejsou žádným způsobem spojité. Práce s mračnem při vizualizaci je obtížná z důvodu neplošného vyjádření povrchu snímaného objektu a viditelnost všech bodů uživatele mate. Při proložení rovinou (provádění řezu) mračnem bodů se zobrazují pouze protnuté body. Pro ukládání a vizualizaci je postačující trojrozměrný eukleidovský prostor E^3. V takovém případě je každý bod reprezentován jako uspořádaná trojice (x, y, z), nebo též jako trojsložkový vektor.

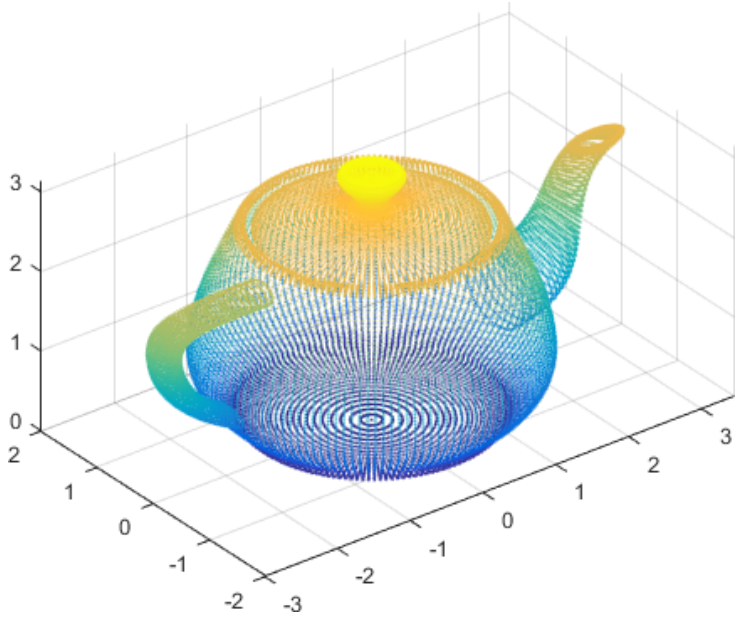
Stejně jako v našem případě jsou mračna bodů vytvářena jako výsledek rekonstrukce 3D skenů. Pak je nutné mračno uložit ve formě datového souboru. Existuje několik užívaných formátů, které umožňují ukládání mračen do souborů. Vybrané dostupné formáty jsou popsány v kapitole Formáty pro uložení mračen bodů.

Aplikace mračen bodů, kromě zmíněného využití ve 3D skenerech, je také v navigaci robotů, konkrétně pro jejich vnímání okolí. Je možné díky nim odhadovat hloubku, zvizualizovat stereo vidění a v pokročilých systémech se využívá pro podporu řízení (například systém *ADAS*). Tento systém poskytuje optimalizované a vysokorychlostní zpracování bodového mračna a jeho následné převzorkování, odstranění šumu a různé transformace pro následné další zpracování dalším software [4].

Ukázka zvizualizovaného mračna bodů, získaného z 3D rekonstrukce vybraného software, je zobrazena na obrázku 1.

2.2 Získání mračen bodů

Při rekonstrukci objektu se využívá promítaní generovaných vzorů s jasnou geometrií. Je jimi do scény (snímku) dodávána pomocná informace sloužící k rozpoznávání. Při osvětlení scény je však projektorem každý snímaný bod osvětlován i nepřímo, a to okolními zdroji světla. Ty však do snímku nepatří, stejně tak jako odražené světlo od okolních osvětlených bodů scény. Je proto zapotřebí se takovémuto osvětlení vyhnout, aby byla rekonstrukce co nejpřesnější.



Obrázek 1: Ukázka vizualizovaného mračna bodů [31]

2.2.1 Promítané vzory

Rekonstrukce objektu je v tomto případě řešení založena na osvětlování projektorem. Vzorové obrazy (byť jen světelné linky či pruhy) musí být vybrány tak, aby bylo možné pro každý bod obrazového prostoru kamery přesně určit, kterým bodem (případně množinou bodů - pruhem) projektoru byl bod osvětlen. Díky tomu jsme pak schopni získat vztah mezi bodem kamery a bodem (případně pruhem) projektoru.

Tyto vzory mohou být rozděleny do několika skupin, od kterých se pak odlišuje následný postup rekonstrukce. Některé jsou určeny na rekonstrukci z jediného snímku, jiné pak z vícero snímků (sekvencí).

První z nich je třída pruhových vzorů, tzv. *fringe patterns*. Jak již název napovídá, tyto vzory jsou specifické tím, že jednotlivé vzorové obrazy sestávají z pruhů. Ty mohou být buďto vodorovné (horizontální) nebo svislé (vertikální). Umístění a směr pruhů však musí být předem známý. Speciálním případem pruhových vzorů je vzor tvořený pouze jednou linkou. Té může být docíleno například projekcí laseru [5].

Pruhové vzory můžeme dále rozdělit do několika podtříd. Jednou z nich jsou vzory založené na **fázovém posunu**. Tyto vzory využívání některou z cyklických matematických funkcí, jakými jsou například *sinus* nebo *cosinus*. Vzor pak vzniká tak, že pro všechny pruhy vypočítáme hodnotu dané funkce, přičemž index (pořadí) pruhu nám slouží jako vstupní parametr. Vzhledem k periodicitě funkce se pak vzor opakuje.

Z pohledu kamery se pak pro každý bod snažíme zjistit jeho funkční hodnotu vzhledem k vysílané funkci a podle ní získat fázový posun funkce pro daný bod vzoru. Pokud víme, ve které periodě se bod nachází, dokážeme určit index pruhu, kterým byl osvětlen. Jsme-li schopni

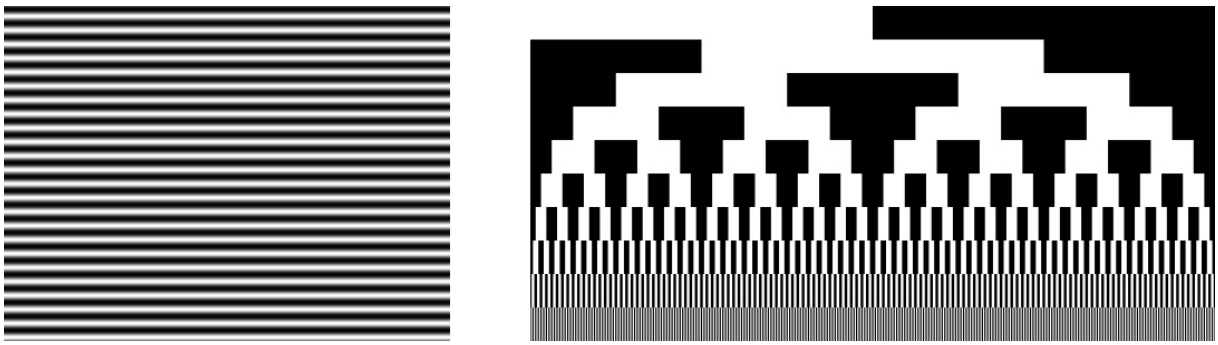
získat tuto informaci z jednoho snímku, je pak teoreticky možné sestrojit model bez použití dalších snímků. Pro zpřesnění rekonstrukce se však používají vícefázové vzory.

Další podtřídou pruhových vzorů je **Grayův zrcadlový binární kód**. Ten je opět složen z horizontálních nebo vertikálních pruhů. Na rozdíl od předchozího typu však musí být každý pruh buď bílý nebo černý. Informace o umístění pruhu je tak rozdělena do několika vzorových obrazů, přičemž každý pruh má unikátní sekvenci zabarvení.

Nechť $n = \lceil \log_2 k \rceil$, kde k je horizontální (resp. vertikální) rozlišení projektoru. Pak i-tý Grayův zrcadlový binární kód se sestává ze série k bílých nebo černých pruhů šířky jednoho pixelu prostupujících obraz projektoru. Přičemž j-tý pruh je bílý právě tehdy, je-li i-tý bit čísla j v Grayově zrcadlovém binárním kódu na hodnotě 1. Jinak je pruh černý [6].

Jednou z hlavních výhod tohoto kódu je, že mezi každými dvěma po sobě jdoucími čísly je rozdíl právě v jednom bitu. Mezi dvěma sousedními pruhy tedy bude právě jednou rozhraní mezi černou a bílou oblastí.

Ukázky obou podtříd pruhových vzorů jsou vidět na obrázku 2. Grayův zrcadlový binární kód je pak zobrazen ve vícero iteracích, přičemž jednotlivé řádky kódu jsou právě námi požadované promítací vzory.



Obrázek 2: Ukázka vzorů (vlevo: fázový vzor, vpravo: Grayův kód)

3 Průzkum trhu a rešerše algoritmů

Tato kapitola shrnuje informace o obdobných systémech vyskytujících se na trhu. Níže uvedené projekty byly odzkoušeny na námi získaných datech. Některé zmíněné projekty jsou dostupné z odkazů, jež jsou uvedeny v závěru této práce v rámci referencí.

3.1 Testovací snímky pro rešerši

Pro testování veřejně dostupných systémů na trhu a následné rešerše algoritmů byla pořízena sada testovacích snímků. Pro snímání byl použit hardware, jehož parametry jsou popsány v části Hardware, uvedené níže v této práci. Prvotní konfigurace kamer a projektoru byla zvolena jako triangulační, kdy stereoskopické kamery (jejichž parametry jsou popsány v první části kapitoly 5.1) byly fixně umístěny na hliníkový profil do téže přímky. Projektor (parametry použitého projektoru jsou popsány na začátku kapitoly 5.2) byl pak umístěn doprostřed mezi ně. Před pořízením datové sady byly kamery nakalibrovány pomocí funkcí knihovny OpenCV, následná kalibrace kamer s projektorem nebyla nutná.

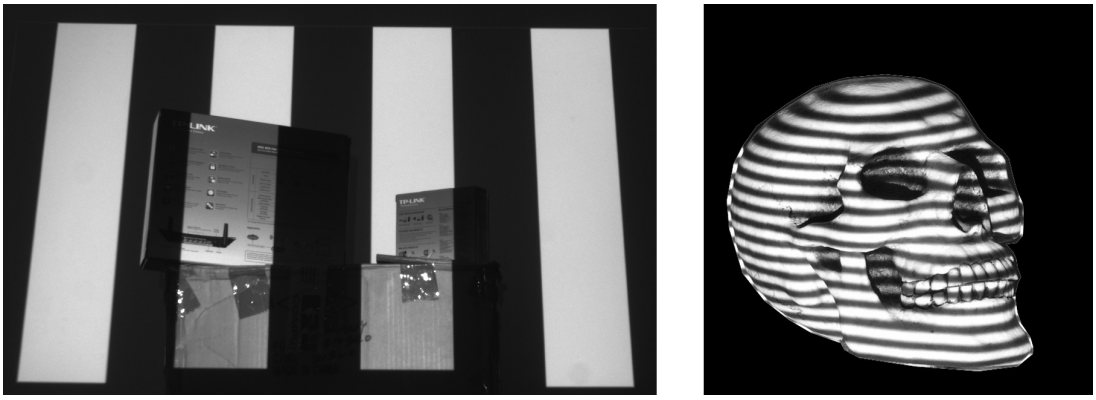
Při pořizování první sady snímků byl projektorem na bílou hladkou zeď promítán vertikální *Grayův kód* (viz kapitola 2.2.1) z vlastní aplikace, sloužící ke generování tohoto kódu. Před takto navrženou konstrukci byly postupně pokládány předměty, které se nejprve osvítily strukturovaným světlem projektoru. Takto získaný obraz byl zaznamenán kamerami a uložen do PNG souboru. Jako snímané objekty byly použity předměty primitivních tvarů, jakými jsou hranoly a krychle. Pro každý objekt je vždy dvojice snímků, získaná z pravé a levé kamery (označených příponami r a l).

Druhá sada snímků již byla pořízena s promítáním třífázových vzorů (viz kapitola 2.2.1). Byla rovněž pozměněna konfigurace kamery a projektoru, jelikož tato sada byla pořízena v domácím prostředí. Nevyužívaly se již stereoskopické kamery jako při pořizování první sady snímků, byla již využita pouze jedna kamera (popsána v druhé části kapitoly 5.1). Kamera a projektor byly v tomto případě umístěny fixně do přímky. Snímaná tělesa byla umisťována před tuto konstrukci a byla postupně osvětlována promítanými vzory.

Takto získané sady snímků byla ukládány do archivů pro následné využití a rekonstrukci. Ukázka nasnímaných objektů je vidět na obrázku 3.

3.2 HHSL3DS

Prvním testovaným softwarem je systém *HHSL3DS*. Jedná se o opensource projekt Dánské university založený na 3D skenování objektu za pomoci strukturovaného světla. Systém obsahuje hardwarovou část, která se skládá z nízkonákladových a běžně dostupných zařízení, jako jsou videoprojektor a webkamery, a softwarovou část, obsahující implementaci algoritmů, potřebných k rekonstrukci objektu. Projekt je implementován v programovacím jazyce `Python`.



Obrázek 3: Testovací snímky (vlevo: Grayův kód, vpravo: třífázový vzor)

Základem skenovacího systému je fixní konstrukce, na které jsou umístěny dvě webové kamery a videoprojektor. Jejich umístění tvoří rovnoramenný trojúhelník, kdy kamery tvoří základnu, projektor je pak umístěn ve třetím vrcholu. Kamery a projektor jsou kalibrovány tak, aby snímaly jeden konkrétní bod. Projektor je využíván k promítání světelných vzorů na objekt, které jsou tvořeny horizontálními a vertikálními pruhy bílého světla. Následně je takto osvětlený snímaný objekt zaznamenán pomocí dvojice kamer.

V projektu je využita knihovna OpenCV pro jednodušší práci s obrazem. Na výstupu je pak PLY soubor obsahující (x, y, z) koordináty získaných rozlišujících bodů snímaného objektu. Tyto body tvoří mračno (*point cloud*), které je následně možné otevřít v *Autodesk* systému pro CAD projekty anebo ve veřejně dostupném softwaru *Meshlab*.

Jedinou a zásadní nevýhodou HHSL3DS je, že konfiguraci kamery a projektoru nelze v rámci projektu upravit. Navíc u tohoto projektu zcela chyběl podrobný popis pro správné pořízení snímků objektu (umístění kamer, projektoru a snímaného objektu v prostoru). Aby bylo možné tento software použít pro naše účely, bylo by nutné upravit zdrojové kódy pro konfiguraci a kalibraci. Dalším řešením by bylo uzpůsobit fungování tohoto systému pro předem danou konstrukci skeneru.

V rámci experimentů se nepodařilo zanalyzovat námi získané snímky, software fungoval pouze pro přiložená ilustrační data. Tento problém byl způsoben odlišným pořízením přiložených a námi získaných dat. Z výše zmíněných důvodů nebyl tento projekt zvolen jako výchozí pro tuto práci.

3.3 Three Phase Scanning system

Tento projekt je založen na myšlence, že jednoduchý 3D skener pro rekonstrukci objektu za pomoci strukturovaného světla a zaznamenání kamerami je dostupný pro běžného uživatele. Jedná se spíše o jakýsi návod, jak takovýto skener zkonstruovat a jak objekt správně převést do podoby dat. Základní myšlenka projektu je jednoduchá a zahrnuje čtyři základní body:

- zkompilování samotného projektu (dekodéru),
- nastavení projektoru,
- nafocení snímků,
- následná úprava.

Stejně jako většina obdobných systémů, i je tento založen na triangulaci (výjimkou jsou *time-of-flight* systémy). Triangulační metoda pracuje na základě trigoniometrického principu snímání tří po sobě jdoucích měření trojúhelníků, což je později využíváno namísto zbývajících měření. Pro příklad uveďme snímání malé bílé kuličky z dvou pohledů. Dostaneme tedy dva různé úhly měření. Pokud navíc víme vzdálenost mezi oběma pohledy (kamerami), jsme schopni vypočítat, v jaké vzdálenosti od kamer je kulička umístěna.

Již podle jména tohoto softwaru je zřejmé, že využívá metody třífázového skenování. Ta řeší problém rozmazání promítaného vzoru při oddálení scény od ohniskové roviny projektoru. Jejím principem je promítání tří kosinových vln, konkrétněji jsou to grafy funkcí s předpisy uvedenými níže ve vztahu 1 [8].

$$\begin{aligned} f_1(x) &= \cos(x - 2\pi/3) \\ f_2(x) &= \cos(x) \\ f_3(x) &= \cos(x + 2\pi/3) \end{aligned} \tag{1}$$

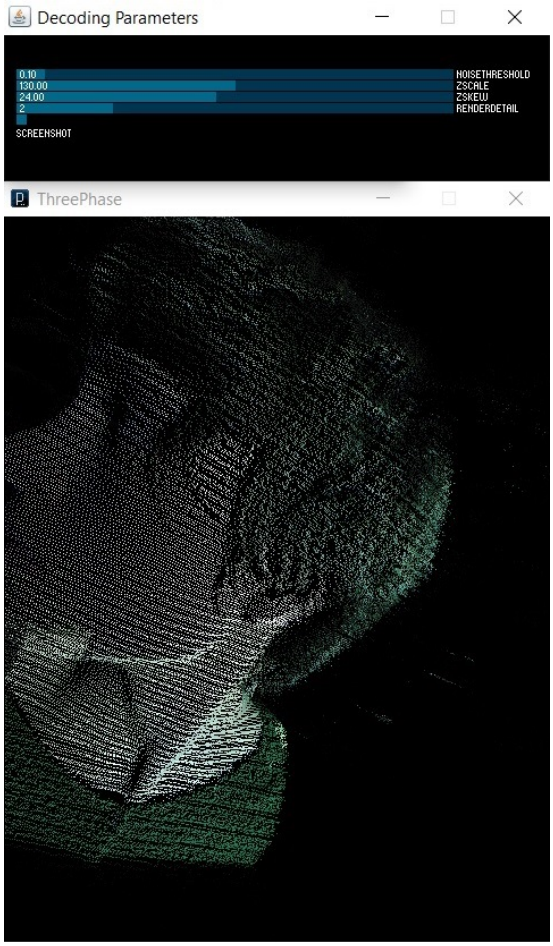
Dále je nutné, aby byl projektor nastaven v režimu na výšku s rozlišením 1024×768 pixelů, a jím promítané vzory byly v rovině ke kamerám. Jinými slovy, aby horizontální čára promítaná projektorem se po nasnímání kamerami skutečně vykreslila jako horizontální čára, čímž jsme se vyhnuli zkreslení obrazu. Snímky musí být zaznamenávány v zatemněné místnosti na bílou hladkou zeď.

Tento systém je v téměř všech bodech shodný s požadavky kladenými touto prací. Stejně jako u výše zmíněného systému HHSL3DS, ani zde autoři neuvádí přesné pozice hardwaru a postup pro správné pořízení testovacích snímků. Problém také nastává u promítaných vzorů tohoto software.

Z tohoto projektu byla pro naši práci převzata pouze výše zmíněná myšlenka pro správné navržení skenovacího zařízení, která byla nadále dodržována. Ukázka fungování Three Phase Scanning systému je na obrázku 4.

3.4 The $30 3D scanner V3 updates

Oproti výše zmíněným projektům se v případě tohoto systému jedná pouze o konstrukci, jak správně získat snímky pro následnou 3D rekonstrukci objektu. Opět je kladen důraz na jednoduchost a běžnou dostupnost. Konstrukce je složena z dílů, jejichž modely jsou dostupné ze stránek

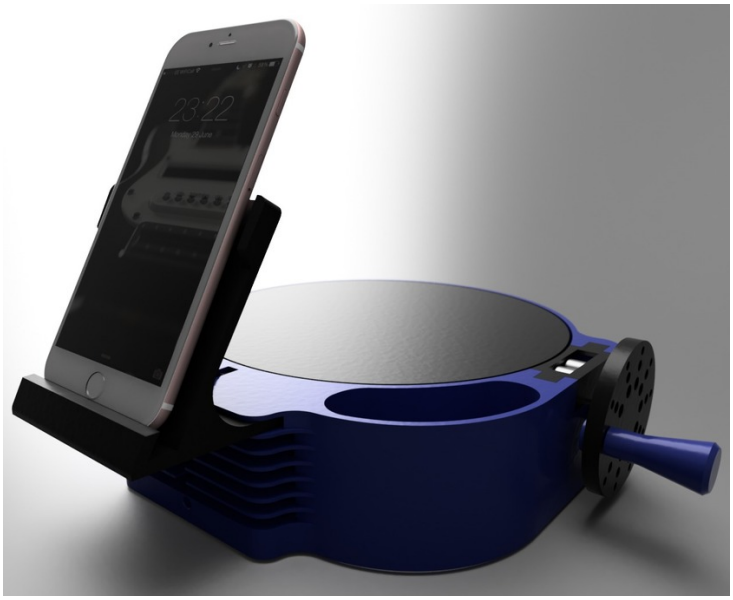


Obrázek 4: Ukázka fungování Three Phase Scanning systému

autora, včetně návodu na jejich složení. Výsledný systém se pak skládá ze tří hlavní komponent: podstavce pro kameru, otočné plošiny pro skenovaný objekt a mechanismu pro otáčení. Roli kamery zde zaujímá mobilní telefon, který je umístěn na příslušný podstavec před snímaný objekt. Otáčení plošiny pak probíhá manuálně za pomoci kliky umístěné na boční straně konstrukce.

Po nasnímání obrázků pak probíhá rekonstrukce objektu. Tuto úlohu zde plní již software třetí strany. Jedná se například o *Autodesk Remake*, který je dostupný i zdarma ve trial verzi. Nutno podotknout, že rekonstrukce již probíhá na počítači, nikoli na používaném mobilním telefonu.

Tento projekt pouze nabízí konstrukci pro snímání objektu. Právě tento nedostatek byl klíčový pro použití výše zmíněných systémů HHSL3DS a Three Phase Scanning system. Tento skener nabízí velmi vhodně zvolené řešení pro umístění jednotlivého hardwaru. Nedostatkem snad může být pouze manuální ovládání rotace objektu. To může způsobovat nepřesnosti v následné rekonstrukci objektu. Ukázka skeneru je vidět na obrázku 5 [9].



Obrázek 5: Ukázka konstrukce 3D skeneru.

3.5 BQ Ciclop

V rámci celkové rešerše a průzkumu trhu byl jako nejvhodnější adept pro rekonstrukci 3D tvaru objektu vybrán systém *Ciclop*. Jedná se o produkt společnosti *BQ* uveřejněný pod mezinárodní licencí *Creative Commons Attribution-ShareAlike* 4.0. Upřesněme, že se jedná o DIY (z anglického *Do it yourself*, tedy *Udělej si sám*) opensource projekt. Skener byl vyvinut na počátku roku 2015 jako projekt v rámci platformy *Kickstarter*, která slouží jako podpora pro kreativní projekty z oblasti technologické inovace, ale také filmu, hudby nebo umění [7].

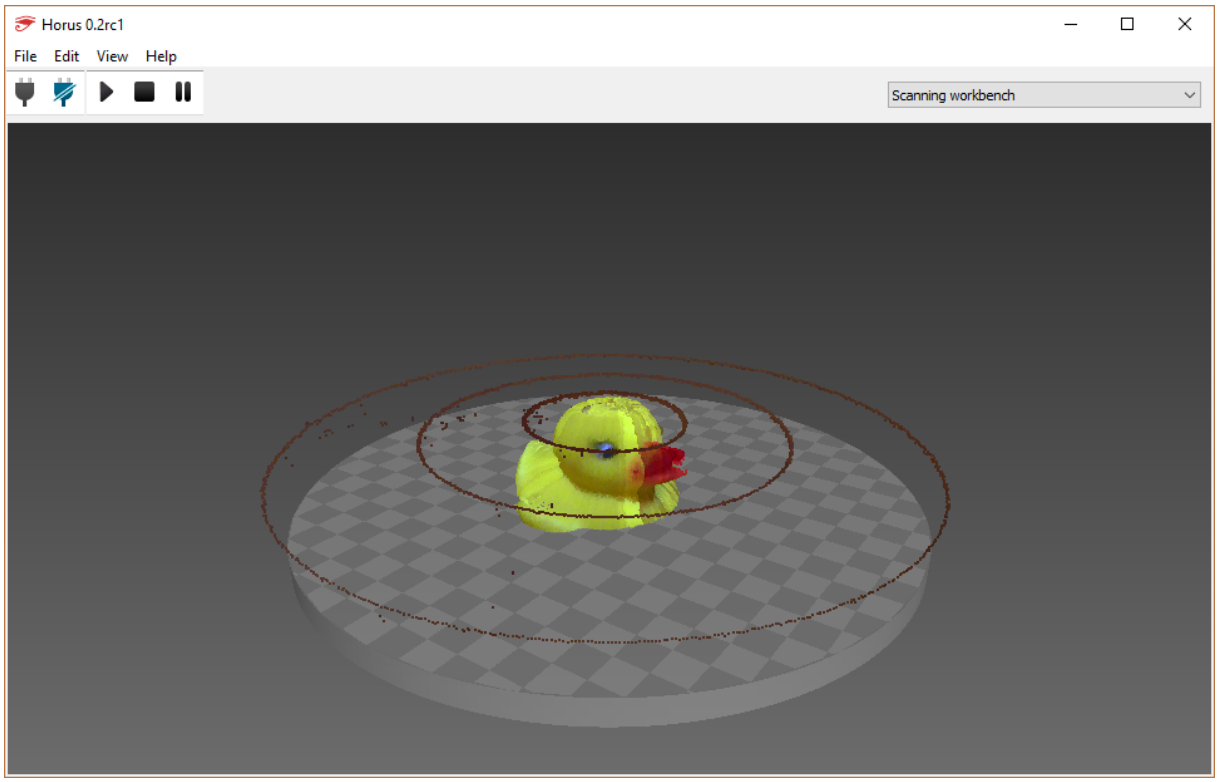
Konstrukce skeneru se skládá z více dílů plnící jasně danou funkci. Jmenujme 3D části pro upevnění laserů nebo otočné plošiny, otočnou plošinu samotnou nebo závitové tyče udržující celkovou stabilitu konstrukce a další (podrobněji v kapitole 5.5). Seznam všech využitých dílů je shrnut do BOM dokumentu a je dostupný z odkazů v referencích. V konstrukci je rovněž i zakomponován hardware sloužící jako řídící jednotka celého systému, projektor strukturovaného světla či snímací zařízení (kamera).

Základem celého systému je již zmíněná otočná platforma. Ta je reprezentována kruhovou deskou z libovolného materiálu, není tedy součástí projektu jako takového, záleží na daném uživateli, jaký typ plošiny zvolí. V rámci specifikace však autoři doporučují metylakrylátovou pogumovanou plošinu, z důvodu stability skenovaného objektu. Podotkněme rovněž, že se jedná o laserový 3D skenovací systém. Rozdíl je zřejmý, projekce strukturovaného světla je zajišťována pomocí laserů [10, 11, 12].

Hardwarová část skeneru již byla zmíněna (a dále ještě bude), přejděme tedy k části softwarové. Ta je reprezentována samostatnou knihovnou Horus (více v kapitole 4.3), která zahrnuje celou ovládací a analýzovou činnost skeneru.

Systém BQ Ciclop byl zprovozněn a řádně otestován. Po jeho analýze byl zvolen jako výchozí projekt pro tuto práci. Toto však neznamená, že byl ponechán v původním, veřejně dostupném stavu. Naopak, celý systém byl podrobně rozebrán, byly zpracovány jeho hlavní klady a zápory, které se týkaly jak hardwarových částí, tak firmwaru a softwaru. Ze skeneru společnosti BQ byla tedy převzata základní myšlenka a konstrukce, zbývající části však byly od základu přepracovány (dále v kapitole 6). Jelikož je tento systém publikován pod výše uvedenou licencí, bylo toto vše uskutečnitelné. A právě z tohoto důvodu nebyly v této kapitole podrobně rozebrány a popsány všechny jeho součásti. Byly zahrnuty pouze ty části, které jsou pro BQ Ciclop stěžejní, zbývající myšlenky jsou popsány v rámci kapitol 4.3, 5.1, 5.3 a 5.5 této práce, jelikož je vhodné je pro přehled popsat v samostatných kapitolách.

Ukázka fungování tohoto systému je vidět na obrázku 6. Poznamenejme, že je zde znázorněna funkčnost původního software bez jakýchkoli dalších úprav.



Obrázek 6: Ukázka fungování BQ Ciclop systému

4 Software

V rámci této kapitoly se seznamujeme s veškerým programovým vybavením využívaným v této práci. Jsou podrobně popsány zvolené programovací jazyky včetně jejich využití v daných částech práce, dále pak využívaná vývojová prostředí a popis knihoven třetích stran, které jsou v práci zakomponovány především pro usnadnění zpracování obrazu či kvůli hardwarové komunikaci. Součástí kapitoly jsou rovněž i využívané formáty pro uložení mračna bodů.

4.1 Volba programovacího jazyka

Celkové řešení této práce zabíhá do zcela odlišných částí informatiky. Předním omezením byl výpočetní výkon a velikost jeho úložiště. Právě z těchto důvodů byla volba vhodného programovacího jazyka obtížná, a ve výsledku jich bylo použito hned několik ve vzájemné spolupráci.

Existuje mnoho kandidátů z řad programovacích jazyků, které připadaly v úvahu jako primární jazyk. Jako první zmiňme jazyk `C#` společně s frameworkem `.NET`. Jedná se o vysokoúrovňový objektově orientovaný jazyk vyvinutý firmou *Microsoft*. K použití jazyka je potřeba framework, který zabírá příliš mnoho místa na disku a zpracování by bylo pro potřeby výsledku příliš pomalé. Z toho důvodu nebyl jazyk použit.

Programovací jazyk `Java` je stejně jako výše uvedený jazyk objektový, jeho běh může probíhat na libovolném zařízení, které má k dispozici virtuální stroj (*Java Virtual Machine*). Technologie `Java` byla taktéž vyloučena z důvodu výpočetních a paměťových nároků.

Možnou volbou byl i jazyk `C++`. Kód napsaný v tomto jazyce je těžko udržovatelný, základní knihovny (například *collection framework*) jsou ve srovnání s moderními jazyky těžkopádné. Tento nedostatek je vylepšován v novějších verzích jazyka (například `C++11` nebo `C++14`), jejichž podpora u různých překladačů není stoprocentní. Variantou je kombinace s knihovnou `Boost`, která slouží k usnadnění programování některých úloh v jazyce `C++`. Její použití by však celé řešení velice zkomplikovala, a tudíž bylo od tohoto jazyka upuštěno [13, 14].

4.1.1 Programovací jazyk `Go`

Jedná se o kompilovaný multiparadigmatický programovací jazyk od společnosti *Google*. Přestože je ze strany `Go` zaručena typová bezpečnost, jsou v něm obsaženy i vlastnosti typické spíše pro dynamicky typované jazyky. V této práci je jazyku `Go` věnována samostatná podkapitola, aby čtenáři byly představeny jeho klíčové vlastnosti, některá zajímavá řešení, jimiž jazyk disponuje. Pro vyzkoušení jazyka není nutná instalace jeho překladače, na webových stránkách je možné vyzkoušet si interaktivní online kurz [15].

Hlavním důvodem, proč byl tento jazyk použit, byla především snadná udržovatelnost kódu, multiplatformnost, a také jeho jednoduchost. Všechny jeho klíčové vlastnosti byly pro tuto práci výhodné, jelikož se jedná o projekt zasahující od komunikace s hardware a síťové komunikace po zpracování obrazu.

```
1  func swap(x, y string) (string, string) { return y, x }
2
3  if _, err := io.Copy(x, y); err != nil { return err }
4
5  var x, y int
6  sum := 0
7
8  type Abser interface { Abs() float64 }
9
10 defer file.Close()
11
12 go f(x, y, z)
```

Výpis 1: Ukázka kódu jazyka GO

Jako první zmiňme syntaxi jazyka `Go`. Ta se od ostatních běžně používaných jazyků liší v několika ohledech. Stejně jako jazyk `Python` se jednotlivé příkazy neukončují symbolem středník. Přesto však existuje několik výjimek, u kterých je použití středníku nutné. Typickým příkladem je například konstrukce podmínky, kdy je možné návratovou hodnotu dané funkce ihned otestovat (řádek 3 ve výpisu 1). Další syntaktickou zajímavostí je striktní pojmenovávání členských proměnných a funkcí, kdy velké počáteční písmeno vždy označuje veřejný přístup, naopak malé počáteční písmeno značí privátní přístup (řádek 1 ve výpisu 1).

Zajímavá je také možná absence datového typu u definic proměnných, který je určován z pravé strany výrazu (řádek 6 ve výpisu 1). Proměnná se inicializuje symboly `:=`, obdobně jako v jazyce `Pascal`. Je však možné překladači jednoznačně datový typ určit, přičemž je nutné proměnnou uvést klíčovým slovem `var` a datový typ je uveden až jako druhý v pořadí (řádek 5 ve výpisu 1).

Jazyk `Go` také umožňuje deklarovat funkce s více návratovými hodnotami, jejichž pořadí je však závazné (řádek 1 ve výpisu 1). To je u většiny programovacích jazyků možné pouze za použití struktury jako návratové hodnoty. Této vymoženosti se hojně využívá hlavně u funkcí, během jejichž vykonávání může nastat chyba. Rovněž zde je možné využít znaku podtržítka na dané pozici jedné z návratových hodnot, které naznačuje, že tato hodnota pro nás není důležitá a nebude tak dále držena v paměti (řádek 5 ve výpisu 1).

Dalším zajímavým prvkem tohoto jazyka je klíčové slovo `defer`. Funkce (ať už anonymní, tedy s tělem definovaným přímo na místě, nebo její volání pomocí identifikátoru) s tím označením je volána až po ukončení funkce, ve které je umístěna. Můžeme si ji představit jako ekvivalent bloku `finally` v jazyce `Java`. Nejčastěji se takto označená funkce stará o uzavření všech otevřených proudů nebo síťových spojení, či odemčení atomických proměnných. Rovněž je doporučeno, aby volání `defer` funkce následovalo ihned po otevření, nikoli až na konci definice

těla dané funkce, jak tomu bývá zvykem u většiny jazyků. Její syntaxi můžeme vidět na řádku 10 ve výpisu 1.

Zajímavým způsobem jsou v `Go` řešeny rozhraní. Je zde využíváno takzvaného *duck typingu*. To znamená, že pokud třída plní daný předpis (řádek 8 ve výpisu 1), tedy pokud definuje veškeré funkce, jejichž předpis je uveden v daném rozhraní, automaticky takovéto rozhraní implementuje a není nutné použití jakýchkoli klíčových slov.

Paralelismus je v tomto jazyce řešen velmi jednoduše, a to použitím klíčového slova `go`. Jakákoli funkce, jejíž volání následuje po tomto klíčovém slově implicitně běží v paralelním vlákně. Ukázka asynchronního volání funkce je vidět na řádku 12 ve výpisu 1.

K výpisu níže je nutné poznamenat, že v něm jsou uvedeny pouze určité výseče kódu, a slouží jen jako ukázka vlastností popsaných výše. Není tudíž není spustitelný.

4.1.2 Sekundární programovací jazyky

Jak již bylo zmíněno v úvodu této kapitoly, vedle jazyka `Go` bylo využito i dalších jazyků. To bylo zapříčiněno návrhem celého systému, jelikož se jedná o aplikaci **server-klient**, kde klientská část je reprezentována webovým uživatelským rozhraním, přes které je celý systém ovládán. Pro implementaci tohoto rozhraní byl využit značkovací jazyk `HTML`. V době psaní této práce byl dostupný ve stabilní verzi 5. Ta oproti dřívějším verzím nabízí prostředky pro využití multimédií, které byly v minulosti řešeny přídavnými pluginy. Navíc, a to je pro tuto práci nezbytnou výhodou, již obsahuje prvek plátna, který umožňuje přímo v rámci stránky interpretovat vektorovou grafiku [16].

V kombinaci s `HTML5` je rovněž využíván i `JavaScript`, který zde slouží pro dodání logiky všem prvkům webového rozhraní nebo k odchytávání a zpracovávání akcí od uživatele. Poznamenejme, že skripty jsou umístěny v samostatných souborech, nikoli jako přímá součást kódu webové stránky.

Pro komunikaci s použitým hardware bylo dále nutné využít jeden s nižších programovacích jazyků. Ačkoli je na této úrovni i zmíněný jazyk `Go`, vhodnějším kandidátem pro toto řešení byl jazyk `C`, zde používaný ve verzi 11. Jeho hlavní výhodou je přímá práce s pamětí (ukazatele nejen na data, ale i na funkce). Z toho vyplývá vysoká výpočetní rychlost, avšak na úkor případného zhroucení programu při nesprávném využití ukazatelů, jelikož zásahy do operační paměti počítače jsou zcela v režii programátora [17].

Pro vytváření balíčků v jazyce `Go`, které provolávají kód psaný v jazyce `C`, existuje příkaz `cgo`. Postačí naimportovat pseudobalíček `C`, díky kterému je pak možné provolávat jednotlivé součásti jazyka `C`. Jako příklad uveďme `C.size_t`, `C.stdout` nebo `C.putchar`. Navíc, pokud se před importem nachází komentář, je nazván preambulí, a může obsahovat libovolný kód napsaný v jazyce `C`, včetně deklarací a definic funkcí a proměnných. Takovýto kód je pak možné přes jazyk `Go` volat [18].

Na závěr doplňme jednu zajímavost, a to, že primárně zvolený jazyk `Go` a výše popsaný jazyk `C` je dílem týž autorů.

4.2 Vývojová prostředí

S ohledem na použití několika programovacích jazyků bylo využito vícero vývojových prostředí v závislosti na jazyce a platformě. Celý vývoj probíhal pod operačním systémem Linux, což byla i cílová platforma celého řešení, jelikož operační systém řídící jednotky skeneru je založen právě na Linuxovém jádře. I přes toto byl stále dodržován důraz na multiplatformnost, protože i použitý hardware umožňuje instalaci operačního systému na bázi Windows. Přestože nebyla tato možnost využita, výsledný řídící software je rovněž spustitelný i na této platformě.

Pro zvolený primární programovací jazyk `Go` byly v době tvorby této práce dostupné nástroje pro tvorbu jeho kódu (například *Gogland* nebo plugin *GoClipse* do nástoje *Eclipse*). Jelikož je jazyk velmi jednoduchý, pro psaní kódu plně poslouží i běžně dostupný textový editor. V našem případě vývoj probíhal v programu `gedit`, jenž dokonce nabízí zvýrazňování syntaxe jazyka `Go`. Kompilace pak probíhala ručně přes příkazovou řádku. Stejného nástroje bylo využíváno i při implementaci webového klientského rozhraní.

Při vytváření rozhraní pro hardware byla implementace rozdělena do dvou větví. Na systému Linux byl využit opět program `gedit` a sestavování skrze `Makefile` s využitím překladače `g++`. Naopak v případě implementace na systému Windows se o překlad již staral nástroj *Visual Studio 2015* ve verzi *Professional*. Jeho hlavní výhodou je usnadnění implementace za pomoci automatického doplňování kódu či našeptávače syntaxe.

Na závěr poznamenejme, že ačkoli byl cílový operační systém *Raspbian*, vývoj a testování probíhalo nad linuxovou distribucí *Xubuntu*. Dále bylo užíváno 64bitového operačního systému *Windows 10* verze *Professional*.

4.3 Knihovny třetích stran

V rámci vývoje skenovacího systému bylo využíváno několik knihoven třetích stran. Jedná se především o knihovny pro práci s obrazem nebo komunikaci s hardwarovým vybavením. Tyto knihovny již mají algoritmy optimalizované na velmi vysokou úroveň, a proto bylo záhodno je alespoň částečně využít.

4.3.1 OpenCV

Jedna z existujících knihoven pro práci s obrazem je *OpenCV*. Jedná se o svobodnou a opensource multiplatformní knihovnu společnosti *Intel*. Knihovna nabízí širokou škálu funkcí a vymožeností pro manipulaci s obrazem. Dokonce obsahuje i metody pro přístup k hardware, v našem případě bylo vhodné specifikovat tuto vlastnost na práci s kamerou. Velkou výhodou této knihovny je její rozsáhlá a přesná dokumentace. Mimo to existuje mnoho návodů a přesných postupů řešení daných problémů i z řad uživatelů. Další velkou výhodou je již zmíněná multiplatformnost celé knihovny [20].

V první fázi vlastního řešení byla knihovna plně využívána především pro komunikaci s kamerou. Během testování se však ukázalo několik nedostatků. Ačkoli knihovna umožňuje přístup

k více kamerám, uživatel nemá informaci o jejich názvech (identifikátorech), takže v případě práce s konkrétní kamerou je nutné znát její systémový identifikátor. Dále pak není možné vylistování všech připojeným snímacích zařízení.

Další, stěžejní nevýhodou, je velikost celé knihovny. Vlastní řešení dbá na zachování jednoduchosti a co nejmenší prostorové a výpočetní náročnosti. Velikost veškerých zdrojových kódů této knihovny je v řádech stovek megabajtů, což je vzhledem k velikosti úložiště mikropočítače Raspberry Pi 3 (Model B), které je závislé na velikosti vložené paměťové karty, neřešitelný problém.

I přes toto byla knihovna OpenCV v práci částečně ponechána k účelům kompilace pod platformou Windows. Byly však naimplementovány vlastní algoritmy pro práci s obrazem, u kterých byl kladen důraz na vysokou optimalizovanost. Setrvávající problém však byla komunikace s použitou kamerou na té nejnižší výpočetní úrovni. K těmto účelům však byla využita jiná, obdobná knihovna popsaná níže v kapitole 4.3.2.

4.3.2 Video4Linux verze 2

Celý vývoj algoritmů pro rekonstrukci tvaru objektu probíhal především pod systémem Linux. Zatímco výše zmíněná knihovna OpenCV nabízela multiplatformní přístup (to znamená plnou podporu systému Linux i Windows), tato knihovna je vytvořena přímo pro Linux. Knihovna poskytuje ucelené API pro zachytávání výstupů (videa) z nižších rozhraní a ovladačů [21].

Po odklonění vývoje od knihovny OpenCV, byla pro práci s kamerou využívána právě knihovna *Video4Linux*. Práce s kamerou přes tuto knihovnu je jednoduchá a možnosti knihovny splňují naše požadavky ve všech bodech. Navíc velikost jejich zdrojových kódů jsou ve srovnání s knihovnou OpenCV minimální.

Při kompilaci projektu na platformu Windows není možné Video4Linux využít. Verze pro Windows tedy stále využívá řešení obsahující knihovnu OpenCV. Jelikož je výsledné řešení primárně určeno pro Linux, nejedná se o zásadní problém.

4.3.3 Horus 3D Scanner Firmware

Knihovna Horus nabízí komplexní řešení pro laserové skenování 3D objektů. Poskytuje grafické uživatelské rozhraní pro připojení, konfiguraci, ovládání, kalibraci a skenování skeneru BQ Ciclop. Tento projekt využívá *Horus 3D Scanner Firmware*. Jedná se o firmware implementovaný v jazyce `C` a vychází z firmwaru *GRBL*. Právě ten je uzpůsoben pro ovládání krokového motoru přes Arduino (a v rámci této práce byl rovněž odzkoušen). Horus 3D Scanner Firmware je navržen tak, kromě krokového motoru byl schopen řídit i činnost laserových modulů.

Komunikační protokol k ovládání laserů a krokového motoru byl pro výsledný skenovací systém využit. Zdrojové kódy Horus firmwaru byly pro účely této práce upraveny z důvodu překonfigurování jednotlivých pinů Arduina, jelikož původní zdrojové kódy byly nastaveny pro odlišné rozpoložení.

4.4 Formáty pro uložení mračen bodů

Jelikož výstupem výsledného řešení má být již připravené mračno bodů, je potřeba ho předat dál pro další zpracování. Existuje celá řada možností, jak mračno reprezentovat a uchovat. V této kapitole jsou popsány jak ty nejběžnější formáty pro uložení mračen, tak i vlastní vytvořený formát, sloužící pro interní způsob ukládání a předávání výsledného mračna bodů.

4.4.1 Formát PLY

Tento formát je dílem Stanfordské univerzity a je koncipován především pro trojdimenzionální data. Slouží tedy pro ukládání objektů ve formě kolekce polygonů (nejčastěji trojúhelníků), které jej tvoří. Formát PLY podporuje uložení ve formě textu i binárních dat. Pořadí obsahu v souboru je následující:

- hlavička s metadaty,
- seznam vrcholů,
- seznam povrchů,
- volitelně další seznamy prvků.

Díky možnostem rozšíření lze uložit i informaci o povrchu, použitých vlastnostech pro shadery a podobně. Ukázka formátu je zobrazena na výpisu 2 [23].

```
ply
format ascii 1.0
comment made by Greg Turk
comment this file is a cube
element vertex 8
property float x
property float y
property float z
element face 6
property list uchar int vertex_index
end_header
0 0 0
0 0 1
0 1 1
// another coordinates of object polygons following
```

Výpis 2: Ukázka formátu PLY (převzato z [23])

4.4.2 Formát OBJ

Dalším běžně používaným formátem je OBJ a k němu přidružený MTL. Formát MTL popisuje materiál povrchů, což pro většinu mračen není potřeba. Samotný OBJ je textový formát, kdy každý řádek obsahuje jeden příkaz. Formát neobsahuje žádnou hlavičku (lze použít vlastní komentář začínající znakem `#`). Nejjednodušší způsob uložení vrcholů je vidět na výpisu 3 [24].

```
# OBJ soubor
v 0 0 0
v 0.5 0 1.0
v 1 2 3.5
v 5 7.5 7
```

Výpis 3: Ukázka formátu OBJ

4.4.3 Formát PPP

Zatímco výše zmíněné formáty jsou běžné a podporované většinou nástrojů, formát PPP vznikl pro účely této práce a je interně používán výsledným skenerem pro uchování mračen bodů. Formát PPP se zpravidla ukládá do souborů s `.ppp` příponou.

Data jsou uchovávána v binární podobě, ale formát nabízí i variantu ve formě kódování Base 64, takže může být přenášen i jako text. Na začátku souboru je hlavička o délce nejméně 32 znaků (v tomto případě bajtů) v podobě `application/x-pstros-ppp-cloud`, zakončená znaky s kódem 13 a 10, představující zakončení řádku (započítávají se do hlavičky). Podoba hlavičky je pevně daná a zakončení pouze pomocí znaku s kódem 13, nebo naopak pouze pomocí znaku s kódem 10, není přípustné. Volitelně lze formát uložit pomocí kódování Base 64, v takovém případě je hlavička `application/x-pstros-ppp-cloud;encoding=base64` a opět je zakončena znaky s kódem 13 a 10. Hlavička nesmí obsahovat žádné mezery ani volitelné části navíc a musí být v jednom z výše uvedených tvarů, aby bylo možné její rozpoznání pomocí porovnání řetězců. Obsah hlavičky odpovídá platnému formátu MIME podle RFC 2046.

Bezprostředně za koncem hlavičky (znak s kódem 10) je uveden počet bodů uloženého mračna. Všechny binárně uchovávané hodnoty jsou zapisovány v malé endianitě (*little endian*). Pro binární formát je počet uložen v 64bitovém čísle bez znaménka, přičemž nejvýznamnější bit je vždy nulový. Maximální počet bodů je tedy v rozsahu 0 až $2^{63} - 1$, přičemž nulový počet bodů je přípustný. Pro variantu využívající kódování Base 64 je počet bodů uchován jako dekadické číslo ve formě řetězce. Obsahovat může pouze číslice (znaky s kódem 48 až 57 včetně), znaménko nebo nuly na začátku čísla, včetně mezer mezi číslicemi a podobně, nejsou přípustné. Maximální počet číslic je 18, maximální hodnota vyjádřená číslem je $2^{63} - 1$, přičemž nulový počet bodů je přípustný. Pro tuto variantu je číslo zakončeno posloupností znaků s kódem 13 a 10, která zde musí být přesně v tomto tvaru a nelze ji vynechat.

Tabulka 1: Uložení bodu v binární verzi formátu *PPP*

X				**Y**				**Z**				**R**	**G**	**B**	**A**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Zbylá část souboru jsou data, která reprezentují jednotlivé body mračna. V binární formě zabírá každý bod přesně 16 bajtů. Čísla jsou opět v malé endianitě a jejich rozložení je vidět v tabulce 1. Každá souřadnice bodu (X, Y, Z) je reprezentována 4bajtovým číslem s pohyblivou řádovou čárkou (IEEE 754, single/binary32). Barevné složky bodu (R, G, B, A) jsou reprezentovány vždy jedním bajtem bez znaménka.

Pro Base 64 je formát dat bodů mračna totožný a po uložení jsou data zakódována pomocí Base 64, jak popisuje RFC 4648. Za touto částí se v obou případech (binární i Base 64 varianta) nesmí nacházet žádná data navíc.

5 Hardware

Po seznámení se s programovým vybavením přichází na řadu fyzická (hardwarová) část práce. Její součástí jsou součástky využívané jak při rešerši a testování existujících obdobných projektů na trhu, tak při výsledném řešení.

Celý systém pro rekonstrukci 3D tvaru objektu nemá kladeny přílišné nároky na použitý hardware. Jak již bylo zmíněno v úvodu této práce, a rovněž i v kapitole Three Phase Scanning system, záměrem je, aby byl výsledný systém dostupný běžnému uživateli. Bylo tedy nutné zachovat tento záměr, a neklást tudíž přílišné výkonnostní a finanční nároky k sestavení koncepce celého systému.

5.1 Kamera

Při celém průběhu řešení této práce bylo použito několik druhů kamer. Je tomu tak z důvodu, že při rešerši stávajících systému se používaly odlišné kamery. Daný software měl buďto striktně zadaný druh kamery (např. Ciclop, popsaný v kapitole 3.5) anebo typ kamery neodpovídal obecným požadavkům systému.

Celkově bylo využito tří typů kamer. Jejich základní a klíčové vlastnosti zachycuje níže uvedená tabulka 2.

Tabulka 2: Hardwarová specifikace použitých kamer

Kamery			
název:	**JAI CM-140GE**	**Toshiba Camileo Pro HD**	**Logitech C270**
specifikace:	čb progresivní kamera	digitální videokamera	webová kamera
rozlišení:	1380 px × 1040 px	HD	HD
snímání:	31 snímků/sekunda	30 snímků/sekunda	30 snímků/sekunda
rozhraní:	programovatelné GPIO	2,5” TFT LCD displej	Logitech Vid HD
další:	GigE Vision interface	3x optický zoom	Logitech RightLight

Dále poznamenejme, že první zmíněná kamera *JAI CM-140GE* byla použita při pořizování testovacích snímků pro rešerši, a byla zastoupena stereoskopicky (tedy při pořizování snímků byly využity dva kusy). Druhá zmíněná *Toshiba Camileo Pro HD* byla použita při snímání druhé sady testovacích snímků pro rešerši, a dále jako kamera při testování systému Three Phase Scanning system. Poslední popsaná kamera *Logitech C270* je pak zvolena jako výchozí kamera pro systém BQ Ciclop, jelikož jej výrobce přímo uvádí v BOM dokumentu součástí, přikládaném k tomuto software. Tato kamera byla i dále používaná během celého vývoje vlastního skeneru a je uváděna jako výchozí. Rovněž i veškeré testování výsledného řešení probíhalo právě s využitím tohoto typu kamery.

5.2 Projektor

Rekonstrukce 3D tvaru objektu je řešena s využitím strukturovaného světla, a to je promítáno právě projektorem. Stejně jako v případě kamer, i projektorů bylo v rámci celé práce využito vícero typů, rozdílný pro rešerši a dále pak pro výsledné řešení.

Během rešerše stávajících systémů byl využit projektor zapůjčený z VŠB-TUO. Ten byl použit při pořizování jak první, tak i druhé sady snímků. Jedná se o běžně dostupný projektor s pro nás dostačujícími parametry. Jeho specifikace je uvedena v tabulce 3.

Tabulka 3: Hardwarová specifikace použitého projektoru

Projektor	
název:	**BenQ MX511**
specifikace:	DLP projektor
rozlišení:	XGA (1024 px × 768 px)
kontrast:	3000 : 1
rozhraní:	HDMI
další:	3D projekce DLP Link

Při volbě konečného řešení této práce bylo pomýšleno na projektor v jiném smyslu. Jelikož projektor jako takový musí plnit jediný účel, a to projekci námi vybraných vzorů. Vše pochopitelně v kvalitě, která musí odpovídat vysoké kvalitě z důvodu následné analýzy obrazu s osvětlovaným objektem. Toto vše ale v základu plní i bodové nebo linkové **lasery**. V jejich případě jsme omezeni jediným aspektem, a to volbou promítaných vzorů. Jak bylo zmíněno v kapitole 2.2.1, pro automatickou rekonstrukci objektu postačí jednoduché vzory tvořené právě jednou linkou. Této vlastnosti využívá i jeden z testovaných obdobných systémů, konkrétně BQ Ciclop. A jak bude zmíněno dále, právě na tomto systému tato práce staví.

Lasery využívané ve výsledném řešení jsou běžně dostupné součástky. Na konkrétním typu v rámci této práce nezáleží, podmínkou však je projekce rovné linie. Pro rekonstrukci se zde využívá čárových laserů s paprskem červené barvy (3 až 6 V). Tento typ laserů je válcovitého tvaru s průměrem 12 mm a výškou 35 mm a dodává se s 135 mm dlouhým přívodním drátem bez konkrétní koncovky. Ta byla k laseru doinstalována až v rámci konstrukce, viz kapitola 5.5. Volba tohoto typu laseru nebyla náhodná, v BOM dokumentu, přikládaném k systému Ciclop, je právě tento model doporučen a celý systém byl právě s ním řádně otestován.

Dále poznamenejme, že v rámci rešerše algoritmů byly tyto lasery využity dva. Umístěny byly po stranách konstrukce a duálně osvětlovaly skenovaný objekt. Jejich účel byl však pouze z důvodu zpřesnění rekonstrukce. Pro další vývoj byl však již používán pouze jeden laser, jenž byl pro promítání strukturovaného světla dostačující.

5.3 Arduino a CNC Shield

Řídící jednotkou skeneru Ciclop je *Arduino*, konkrétně typ *Uno*. Jedná se o jednodeskový mikropočítač využívající mikrokontrolerech ATmega od firmy Atmel, tento typ je založen na mikrokontroleru ATmega328P. Disponuje 32 KiB flash pamětí, 14 digitálními vstupně-výstupními piny a k počítači jej lze připojit pomocí USB [26].

Úkol tohoto mikropočítače je jasný a jednoduchý, slouží především jako ovládací jednotka pro lasery a krokový motor skeneru. Právě pro takovéto účely je přesně navržen. Poznamenejme, že ovládání webové kamery má již za úkol počítač, ke kterému je toto Arduino fyzicky připojeno.

Původně měl být k Arduinu připojen *ZUM Scan Shield*, který je v kombinaci s Arduinem uzpůsoben právě pro ovládání krokových motorů a laserů. Je obdobou klasického CNC Shieldu. Stejně jako doporučený shield, i tento je obvyklý modul, mající charakteristické rozložení konektorů. Právě díky této vlastnosti není zapotřebí žádného pájení, shield společně s mikropočítačem stačí spojit konektory [27, 28].

5.4 Raspberry Pi 3 (Model B)

Oproti Arduinu je již Raspberry plnohodnotným (byť stále mikro) počítačem. Jedná se produkt společnosti *Raspberry Pi Foundantion* o velikosti běžné platební karty. Raspberry je taktéž jednočipový počítač, ovšem se znatelně vyšším výkonem [29].

Jak již název napovídá, jedná se o třetí generaci tohoto mikropočítače, obsahující $1,2$GHz 64-bitový čtyřjádrový procesor rodiny ARM, 1GB operační paměti SDRAM (sdílená s GPU), a co je nejdůležitější, vestavěnou Wifi anténou. Díky procesoru ARMv8 je na Raspberry Pi 3 možné spustit veškeré ARM GNU/Linux distribuce, jakými jsou například *Ubuntu MATE*, *Snappy Ubuntu Core*, nebo M*icrosoft Windows IoT*. Na námi používanou Raspberry byl nainstalován výrobcem distribuovaný operační systém *Raspbian*. Na desce se nachází i zasouvacím microSD slot pro paměťovou kartu. Nadále jsou zachovány stejné rozměry a rozmístění konektorů, jako jeho předchůdci, je tedy plně kompatibilní se všemi stávajícími moduly. Výrobce dokonce garantuje o polovinu vyšší rychlost než Raspberry Pi 2 se stejným software [29].

Díky využití tohoto mikropočítače bylo možné odstínit nutnost připojení stolního počítače a dosáhnout autonomie skeneru. Doposud bylo skenování zcela závislé na výpočetní síle připojeného počítače, veškeré algoritmy pro práci s obrazem, vizualizace a tvorba výsledného mračna bodů byly realizovány pomocí výpočetní jednotky tohoto počítače. S použitím Raspberry však bylo možné přesunout tento výpočetní výkon na její vlastní procesor a osamostatnit tak skener. Toto bylo pro tuto práci stěžejní. Mikropočítač tak pouze po síti, za pomoci komunikačních protokolů, poskytuje uživateli rozhraní pro práci se skenerem, veškerou práci však obstarává sám. Je tedy možné pracovat se skenovacím systém zcela nezávisle. To u výchozího Ciclop skeneru nebylo možné, jelikož vyžadoval fyzické propojení k počítači. V našem případě je toto připojení již bezdrátové.

Poznamenejme dále, že výše zmíněné Arduino s CNC Shieldem nebylo z řešení odstraněno. Je i nadále využíváno k ovládání laserů a krokovacího motoru. Existuje však způsob, jak tyto součástky přímo připojit k Raspberry a zjednodušit tak komunikaci mezi nimi. Je ovšem znovu nutné podotknout, že toho je úloha až budoucích optimalizací, a týká se hardware. Ovládání však není nikterak složitější, oba mikropočítače mezi sebou lehce a rychle komunikují a veškeré ovládání tak probíhá bezchybně. Nevýhodou je prostorová náročnost, jelikož takto musí být součástí skeneru obě tyto jednotky. Nakonec je ještě dobré podotknout, že ovládání kamery již rovněž spadá pod správu Raspberry, jelikož disponuje několika USB porty. Výše zmíněné tvrzení o autonomnosti skenovacího zařízení tak zůstává zachováno. Jediný kabel vedoucí ze skeneru je napájení pro samotný mikropočítač Raspberry Pi 3.

5.5 Konstrukce skeneru

Z důvodu plně automatizovaného skenování a důrazu na kvalitu rekonstrukce objektu bylo důležité vhodně zvolit konstrukci celého skenovacího zařízení. Tím je myšleno patřičné rozložení všech nutných součástí, jakými jsou poloha projektorů strukturovaného světla, umístění kamery, uložení řídících jednotek systému, či fixní plocha pro skenované objekty.

Již během analýzy stávajících systémů se jako odpovídající řešení ukazovalo trojúhelníkové rozložení projektoru, kamery a rekonstruovaného objektu. Všechny testované systémy využívaly právě takovéto idey, pro naše účely se také používá a celý systém je na ni založen.

Tato práce má za úkol rekonstrukci 3D tvaru objektu, proto bylo nutné zvolit takovou konstrukci, která by byla rychle a jednoduše použitelná k výslednému řešení. Systém BQ Ciclop je veřejně dostupný i o se týče hardwarové části. Na stránkách výrobce (viz reference) je možné dohledat celý postup stavby konstrukce. Veškeré díly skenovacího zařízení jsou dostupné ve formátu `.stl` pro běžné 3D tiskárny, ale také v `.fcstd` pro CAD systémy. Jednotlivé součásti tedy nebylo problematické vytisknout, dokoupit další díly, jakými jsou například kuličkové ložisko nebo kovové závitové tyče včetně patřičných matek, a vše sestavit podle přiloženého návodu. Výsledkem tedy je stabilní konstrukce skenovacího zařízení s jasně danými a měřitelnými parametry (obrázek 7).

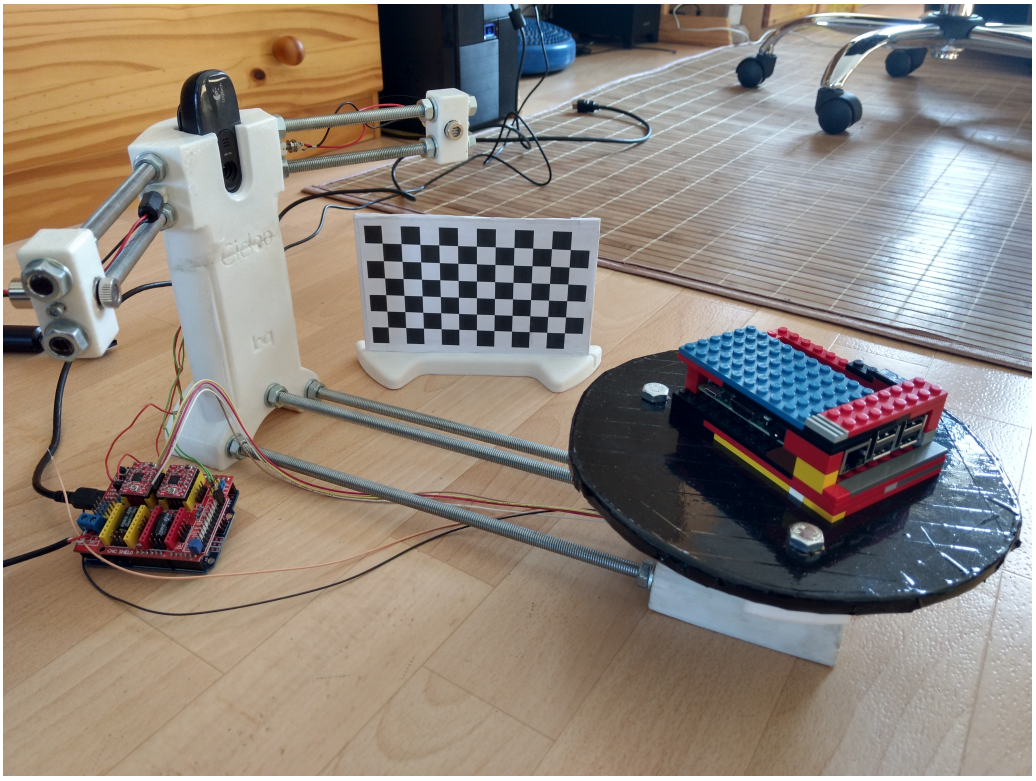
Konstrukce je rozdělena do dvou částí. První z nich je hlavní vertikální věž s prostorem pro řídící jednotku, z jejíž horní části jsou vyvedeny dvě závitové tyče (o délce 170 mm) po obou stranách, na jejichž konci jsou umístěny lasery. V horní části věže uprostřed je dále prostor pro kameru. Z její dolní části jsou rovněž vyvedeny závitové tyče (konkrétně dvě o délce 400 mm, umístěné po stranách, a jedna o délce 292 mm, umístěná ve středu) spojující věž s rotační částí, která reprezentuje druhou hlavní část skeneru. Ta je složena z hlavního stabilizačního dílu, ve kterém je umístěn krokový motor (typ *Nema 17*). Na tento díl je umístěno kuličkové ložisko pro lepší valivý odpor rotační desky o průměru 200 mm, na kterou je později umisťován námi skenovaný objekt.

Prvotně byl skener sestaven přesně dle návodu udávaném společností BQ. Bylo tomu tak z důvodu náležitého otestování systému. Pro námi zvolené řešení se však přesné kopie dále

nevyužívá. Vše bylo ovlivněno právě testováním celé konstrukce, během kterého se ukázaly určité nedostatky popsány dále, anebo námi upřesněnými požadavky. Některé části tedy byly upraveny anebo vylepšeny oproti výrobcem udávanému řešení.

Prvním vylepšením se týká zapojení součástek skeneru, konkrétně samotné zapojení laserů. Zatímco u původní konstrukce byly lasery zapojeny přímo do řídící jednotky (v tomto případě modul Arduino, popsaný v kapitole 5.3), v našem případě jsou z řídící jednotky pouze vyvedeny spoje s koncovkami kompatibilní pro konektor Jack 2,5 mm. Jelikož se použité lasery dodávají bez konkrétních koncovek, nebylo obtížné k vývodům z laseru připájet již zmíněné konektory. Tímto bylo dosaženo větší modularity v případě výměny laserů.

Další problém nastal u rotační plochy pro umístění skenovaného objektu. Výrobce zde nevhodně zvolil řešení nasazení plastové, na 3D tiskárně vytisknuté, násady s průřezem připomínající zaoblené ozubené kolo na osku krokového motoru. Je zřejmé, že v případě jakéhokoli nedostatku při tisku, či při delší době používání se násada na osce začne protáčet, čímž znemožňuje skenování.



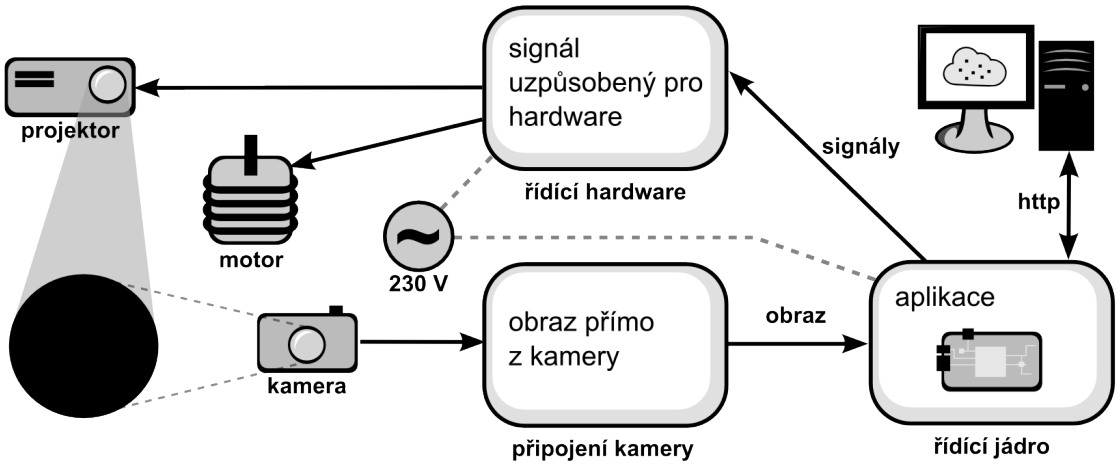
Obrázek 7: Výsledná konstrukce skeneru (včetně hardware)

6 Implementace

Kapitola rozebírá vlastní implementační řešení zvolené pro výsledný produkt této práce. Nahlíží na něj z technického pohledu, a rovněž se hojně odkazuje na již rozebraná teoretická východiska, popsaná ve druhé kapitole této práce.

6.1 Navržené řešení

Z pohledu hardwaru vychází výsledný skener z projektu Ciclop, jak popisují podkapitoly 3.5 a 5.5. Pro rekonstrukci tvaru objektu je potřeba kamera snímající obraz, jeden laser a otočná plošina, na kterou je skenovaný objekt umístěn. Jelikož Ciclop všechny tyto části obsahuje, je možné ho použít jako výchozí hardware k výslednému softwarovému řešení, které je navrženo univerzálně. Součástí skeneru Ciclop je i řídící jednotka (Arduino a CNC shield) s nahraným firmware, se kterým dovede výsledný software pracovat. Rozdělení jednotlivých částí skeneru je vidět na obrázku 8.



Obrázek 8: Schéma fungování skeneru při rekonstrukci 3D tvaru objektu

Projektorem je v tomto případě čárový laser, ale může jít o jakékoliv zařízení, schopné generovat strukturované světlo. *Řídící hardware* je již zmíněné Arduino, které lze v tomto případě nahradit jednodušším zapojením, jak popisuje podkapitola 7.3. Samotný software je spuštěn na *řídícím jádře*, které může představovat počítač s operačním systémem Windows nebo Linux, případně Raspberry Pi 3. Ten musí mít přístup ke *kameře* a *řídícímu hardware*. Uživatel ovládá software skrze uživatelské rozhraní, které v tomto případě představuje webová stránka. Díky tomuto návrhu je možné skenování ovládat i z jiného zařízení, dostupného po síti. Teoreticky je možné, aby jedno *řídící jádro* ovládalo více skenerů, nebo naopak aby více uživatelů pracovalo současně s jedním *řídícím jádrem*. Tento požadavek ale nebyl podstatou této práce, s ohledem na návrh systému však nic nebrání budoucímu přidání této funkcionality.

Softwarové řešení je poté rozděleno do několika větších autonomních celků. Každý z nich je kombinací více programovacích jazyků a plní v roli skeneru specifickou úlohu. Některé z nich lze s ohledem na platformu nebo využívání výsledného systému vynechat nebo zaměnit. Každé z těchto částí je věnována jedna z následujících podkapitol. Jsou zde podrobně rozebrány implementační řešení jednotlivých částí a jejich rozčlenění do balíčků.

Spuštění systému probíhá v několika krocích. Jako první je zaveden projekt P.S.T.R.O.S. (viz kapitola 6.2). Ten automaticky spouští projekt P.S.T.R.O.S.i.C.E (viz kapitola 6.3). Oba následně běží souběžně a komunikují přes vlastní protokol popsaný v kapitole 6.4.

6.2 Projekt P.S.T.R.O.S.

Primárním projektem výsledného řešení je projekt P.S.T.R.O.S. Pokrývá téměř veškeru funkcionalitu skenovacího zařízení, od komunikace s dalšími projekty, ovládání skenovacího zařízení, rekonstrukci 3D tvaru objektu, po implementaci a řízení uživatelského rozhraní. Je kombinací více programovacích jazyků, konkrétně jazyka `Go`, `HTML5` a `Javascriptu`. Projekt P.S.T.R.O.S. je rozdělen do několik balíčků, jejichž účel je uveden níže v této kapitole. Některé balíčky jsou však popsány v rámci samostatných kapitol, jelikož jsou obsáhlé a pro tuto práci stěžejní.

6.2.1 Aktualizační balíček

Součástí projektu P.S.T.R.O.S. je i balíček `pstros/pi/updater`. Ten byl vytvořen z důvodu usnadnění aktualizace nové verze systému na řídící jednotku skeneru (Raspberry Pi 3). Složka balíčku obsahuje dávky (pro Windows i Linux) umožňující sestavení a aktualizaci softwaru ze vzdáleného počítače. Postup aktualizace je následující:

1. `build` provede sestavení celého software pro Linux (ARM7),
2. dojde ke spuštění utility `pstros/pi/up`,
   (a) výsledný spustitelný soubor, složky *templates* a *static* jsou sbaleny do `ZIP` archivu,
   (b) provede se odeslání archivu službě *pstros-updater* na řídící jednotku skeneru,
   (c) aktualizační služba se pokusí spustit novou verzi softwaru, jinak obnoví původní.

Po navázání spojení mezi vzdáleným zařízením provádějícím aktualizaci a cílovou řídící jednotkou skeneru je po dobu aktualizace služba *pstros-updater* uzamčena pro jakékoli jiné akce.

6.2.2 Balíček `cloud`

Tento balíček obsahuje datové struktury a rozhraní pro uchování mračen v paměti a přístupu k jednotlivým bodům. Bod je v tomto případě definován souřadnicemi v prostoru (X, Y, Z) a barvou ve formátu *RGBA*. Základem je propojení s formátem PPP, který je pro projekt nativní.

Jednotlivé podbalíčky, jak je popisuje tabulka 4, pak umožňují nahrávat nebo ukládat běžné formáty pro práci s mračny bodů.

Tabulka 4: Formáty pro nahrávání a ukládání mračen bodů

Balíček	**Název formátu**	**Možnosti**
`pstros/cloud/obj`	Wavefront OBJ	Pouze ukládání.
`pstros/cloud/ply`	PLY	Nahrávání a ukládání.
`pstros/cloud/ppp`	PPP	Nahrávání a ukládání.

Protože formát Wavefront OBJ neumožňuje uložit barvu k bodům, dojde při uložení do tohoto formátu ke ztrátě barevné informace. Formát PLY pak při nahrávání neakceptuje variantu ASCII, což lze v budoucnu doplnit.

Další úlohou balíčku `cloud` je pak získání bodů z obrázku objektu. Obsahuje tedy samotnou implementaci algoritmu pro skenování. Bližší popis je uveden v podkapitole 6.7.

6.2.3 Balíček `impr`

Součástí balíčku jsou jak struktury pro uchování bitmapových obrázků, tak funkce pro jejich zpracování přímo na úrovni jazyka `Go`. Základem je datový typ `Snapshot` který představuje jeden samostatný obrázek v barevném formátu RGB nebo stupních šedi. Barevný formát je uložen ve vlastnosti `Scheme` a nabývá dvou možných hodnot, jak zobrazuje výpis 4. Datový typ `Snapshot` je rovněž kompatibilní s balíčkem `image`, který je základní součástí jazyka `Go`. Pro převod `image.Image` je k dispozici funkce `ToSnapshot`, která přijímá obecné rozhraní a vrací ukazatel na typ `Snapshot`.

```
const (
  Grayscale = iota // 0
  RGB // 1
)

type Snapshot struct {
  Bytes []byte
  Width, Height int
  Scheme int
}
```

Výpis 4: Datový typ `impr.Snapshot`

Funkce pro zpracování obrazu jsou rozčleněny do samostatných souborů v balíčku. Jejich výčet je zobrazen v tabulce 5. V rámci ní jsou funkce popsány z implementačního hlediska, matematická definice stěžejních funkcí, včetně jejich významu ve zpracování obrazu, je uvedena v kapitole 2.

Tabulka 5: Výpis funkcí z balíčku `impr`

Funkce	**Význam**
`DoGauss3x3`	Aplikuje na obraz gaussův operátor ve formě konvoluční masky velikosti 3×3, který způsobí rozmazání obrazu.
`DoGauss5x5`	Obdoba funkce `DoGauss3x3` pro masku velikosti 5×5.
`DoSharpen`	Zaostří obraz pomocí příslušné konvoluční masky velikosti 3×3.
`DoEdgeDetect`	Provede zvýraznění hran v obraze pomocí příslušné konvoluční masky velikosti 3×3.
`DoConvolution`	Zpracuje celý obraz pomocí předané konvoluční masky. Ta musí mít velikost 3×3 nebo 5×5. Parametrem `divider` je poté každý zpracovaný pixel vydělen, pokud není hodnota parametru rovna nule.
`DoDiff`	Odečte od sebe předané obrazy, které musí mít stejný formát. Výstupní obraz je ve stupních šedi.
`DoDilation`	Provede operaci dilatace. Vstupní obraz musí být ve stupních šedi a obsahovat pouze hodnoty 0 a 1.
`DoErosion`	Provede operaci eroze. Vstupní obraz musí být ve stupních šedi a obsahovat pouze hodnoty 0 a 1.
`DoFlip`	Překlopí vstupní obraz o 90 stupňů doprava.
`DoGrayscale`	Převede obraz z barevného do stupní šedi. Všechny složky barvy jsou použity ve stejném poměru.
`DoNaiveThinningInRow`	Provede ztenčení hrany, přičemž každý řádek obrazu je zpracovaný zvlášť.
`DoSobel`	Aplikuje na obraz sobelův operátor.
`DoThinning`	Provede ztenčení hrany pomocí *Stentifordova algoritmu*.
`DoThreshold`	Varianta funkce `DoVariableThreshold` používající pro výsledek hodnoty 0 a 1.
`DoVariableThreshold`	Provede prahování obrazu pomocí hodnoty pevného prahu předaného v parametru funkce. Hodnota výsledku pod prahem i nad prahem je nastavitelná.

Všechny funkce v tabulce pracující s konvoluční maskou pracují i s okrajovými pixely obrázku. Pro potřebu takovéhoto zpracování se hraniční pixely opakují.

6.2.4 Balíček `pstrosice`

Slouží pro komunikaci s projektem P.S.T.R.O.S.i.C.E. skrze síťový protokol popsaný v podkapitole 6.4. Balíček nabízí jednoduché rozhraní pro zprostředkovanou práci se hardwarem skeneru, které je bezpečné pro přístup z více goroutin. Vnitřně používá pro synchronizaci mechanismus mutexů z balíčku `sync`. Balíček rovněž obsahuje proměnnou `Scanner`, která je kompatibilní s rozhraním `Scanner` z balíčku `scanning`.

6.2.5 Balíček `scanning`

Balíček řídí samotný skenovací proces na softwarové úrovni. Přístup ke zdrojům (hardwaru) je řešen skrze rozhraní `Scanner`, které umožňuje:

- zachycení obrázku z kamery (včetně rozdílového obrázku) včetně získání rozlišení,
- oříznout snímaný obraz na menší oblast,
- zapnout a vypnou promítání vzoru,
- otočit skenovaným předmětem.

Skenovací proces je bezpečně přístupný z kterékoliv goroutiny. V průběhu lze vyzvedávat stav procesu včetně již získaných bodů. Body jsou získávány vždy najednou, nikdy nedojde k získání části bodů nebo nekorektních dat. Proces je možné kdykoliv přerušit, přičemž běh se zastaví po dokončení některé ze základních úloh skenování, a to ihned, jakmile to bude možné. Výsledné mračno bodů je uloženo do předem zvoleného souboru, a to bez ohledu na to, zda byl proces skenování dokončen celý nebo byl předčasně ukončen uživatelem.

6.2.6 Balíček `storage`

Balíček umožňuje přístup k úložišti mračen bodů bez ohledu na platformu. Jednotlivá mračna mají názvy odpovídající regulárnímu výrazu: `\w([\w-]\w|\w){0,63}`. Maximální délka názvu je tedy 127 znaků, přičemž název je složen z velkých i malých písmen anglické abecedy a číslic. Může obsahovat i pomlčky, ale ne na začátku, ani na konci nebo dvě za sebou.

Nové jméno je možné vygenerovat pomocí funkce `UnusedName`. Ta vrátí nový název pro dosud neexistující soubor, pokud je jméno předané v parametru již obsazené. Pokud je volána s prázdným řetězcem, vygeneruje nový použitelný název.

Jsou zde obsaženy i funkce pro realizaci výčtu všech mračen včetně podpory stránkování. Soubor mračna je možné otevřít pro čtení i zápis nebo vytvořit nový soubor mračna.

6.2.7 Balíček `virtual`

Balíček poskytuje strukturu `Scanner`, která je kompatibilní se stejnojmenným rozhraním z balíčku `scanning`. Díky tomu lze simulovat hardware skeneru z uložených snímků, pokud není k dispozici hardwarový skener. Volba typu skeneru je možná skrze uživatelské rozhraní, stejně tak lze nastavit cesta ke složce s kolekcemi snímků.

Struktura `Scanner` se skládá z proměnné `Path`, uchovávající cestu ke složce se snímky, a proměnné `Index`, představující číslo aktuálně zpracovávaného snímku. Jednotlivé snímky se musí jmenovat podle vzoru `a`*`X`*`.png`, kde *`X`* představuje číslo snímku (číslováno od nuly včetně). Odečtené snímky (ve stupních šedi) se pak jmenují podle vzoru `b`*`X`*`.png` a korespondují se stejně číslovanými A-snímky. Složka s takto pojmenovanými obrázky se nazývá kolekce.

6.2.8 Uživatelské rozhraní

Ovládání celého softwaru je realizováno pomocí webového rozhraní. Jazyk `Go` v základu poskytuje balíček `net/http`, který umožňuje pracovat se stejnojmenným protokolem z pohledu klienta i serveru. Vytvoření plnohodnotného http serveru vyžaduje tedy jen několik málo řádků kódu, jak je vidět z výpisu 5.

```
package main

import "fmt"
import "net/http"

func main() {
  http.HandleFunc("/", func (w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello World!")
  })
  http.ListenAndServe(":80", nil)
}
```

Výpis 5: Ukázka http serveru v jazyce `Go`

Toto řešení je použito nejen pro realizaci grafického uživatelského rozhraní, ale i pro další části softwaru, jako je API nebo systém pro export mračen bodů (popsáno v podkapitole 6.2.9). Samotné uživatelské rozhraní je pak rozděleno do dvou hlavních částí: *dynamická* a *statická*.

Dynamická část využívá jednotlivé funkce odpovídající datovému typu `Controller`. Název funkce je shodný s adresou URL a šablonou; představuje tedy zpravidla jednu obrazovku v uživatelském rozhraní. O propojení se stará soubor `controllers.go`, který uchovává seznam všech těchto funkcí. Šablony jsou uchovávány ve složce `/templates`.

Statická část je vytvořena pomocí funkce `FileServer` z balíčku `http`, která zprostředkovává soubory ze složky `/static`. V ní jsou uchovány soubory se styly, skripty, písmem, obrázky a další, které jsou potřeba pro sestavení výsledné stránky (obrazovky).

Pro tvorbu klientské části webového uživatelského rozhraní byl použit framework *Bootstrap* ve verzi 3.3.7. Jedná se o sadu stylů a funkcí pro vytváření webových aplikací, založených na `HTML`. Obsahuje mnoho předem připravených řešení pro tvorbu a úpravu formulářů, nabídek, tlačítek a dalších komponent. Jeho výhodou je zkrácení doby vývoje a usnadnění vytvoření graficky přívětivé podoby webu [30].

Na straně serveru je pak pro tvorbu jednotlivých obrazovek použit šablonovací systém jazyka `Go`, obsažený v balíčku `html/template`. Ten nabízí sadu funkcí pro snadné prezentování dat přímo z datových typů jazyka. Základem jsou značky uvozené znaky `{{` a `}}`, které zohledňují kontext a podle toho volí metodu escapování výstupu.

6.2.9 API a export mračen bodů

Ovládání software je možné i skrze poskytované API, které je řešené formou http požadavků. Všechny adresy začínají předponou `/api/` a výsledkem je odpověď ve formátu `JSON`, obsahující ve většině případů objekt. Ten má vždy dvě položky: `done`, obsahující hodnotu `true` nebo `false` podle toho, zda byl požadavek splněn a `error` s případným popisem chyby (nebo hodnotou `null`). Ostatní položky objektu jsou závislé na konkrétním požadavku.

`/api/pstrosice/start`
: Spustí program P.S.T.R.O.S.i.C.E.

`/api/pstrosice/state`
: Zjistí aktuální stav spuštění programu P.S.T.R.O.S.i.C.E. Na výstupu je pouze číslo odpovídající konkrétnímu stavu.

`/api/pstrosice/stop`
: Zastaví program P.S.T.R.O.S.i.C.E.

`/api/pstrosice/laser`
: Zjistí stav laseru. Na výstupu je hodnota `true` nebo `false`.

`/api/pstrosice/log`
: Vyzvedne nově přidané řádky z logu P.S.T.R.O.S.i.C.E. Pole řetězců je v položce `lines`.

`/api/pstrosice/test/laser/on`
: Zapne laser.

`/api/pstrosice/test/laser/off`
: Vypne laser.

`/api/pstrosice/test/camera`
: Zachytí snímek z kamery. Výsledek je ve formátu `image/png` (kódováno jako datová URL v `Base64`) v položce `image`.

`/api/pstrosice/test/motor`
: Pootočí motorem. Parametr `angle` musí obsahovat celé číslo, vyjadřující úhel otočení.

`/api/scanning/autocalibrate`
: Provede vyfocení snímku z aktivního skeneru, přičemž očekává přítomnost kalibračního objektu. V případě, že byl objekt rozpoznán, obsahuje položka `calibration` pole šesti automaticky získaných kalibračních hodnot, položka `image` pak obsahuje obrázek, který je zaslán stejně jako v případě `pstrosice/test/camera`.

`/api/scanning/camera`
: Zachytí snímek z kamery aktuálně používaného skeneru. Výsledek je zaslán stejně jako v případě `pstrosice/test/camera`.

`/api/scanning/cloud`

Výsledkem je část naskenovaného mračna bodů ve formátu PPP. Parametr `from` může být přítomen a vyjadřuje pořadové číslo bodu, od kterého mají být data zaslána. Díky tomu je možné vyzvednout jen nově přidanou část mračna bodů v průběhu skenování.

`/api/scanning/state`

Shrne stav skenování. Položka `pstrosice` obsahuje `true` nebo `false`, v závislosti na stavu spuštění programu P.S.T.R.O.S.i.C.E. Je zde obsažena bez ohledu na to, jaký typ skeneru je vybrán. V položce `scanning` je pak objekt, obsahující položky:

- `active` vyjařující, zda skenování probíhá (`true` nebo `false`),
- `state` zobrazující průběh skenování (číslo v rozmezí 0 až 100 včetně),
- `error` pokud v průběhu skenování došlo k chybě.

`/api/scanning/stop`

Zastavení skenování, které se nemusí zastavit ihned, ale provede se tak co nejdříve.

6.3 Projekt P.S.T.R.O.S.i.C.E.

Jedná se o samostatný program implementovaný v jazyce `C`. Jeho hlavním úkolem je hardwarová komunikace s periferiemi systému, která je závislá na konkrétním operačním systému. Kromě toho však obsahuje i některé funkce pro usnadnění následného skenovacího procesu.

Jak již bylo zmíněno v kapitole 6.1, projekt P.S.T.R.O.S.i.C.E. je automaticky spouštěn při startu celého systému. Mohou nastat celkem čtyři různé situace: **Waiting**, **Running**, **Stopped** a **Failed**. Stav **Waiting** označuje čekání a nastává v případě navazování komunikace s programem. Jakmile je program plně připraven, stav se změní na **Running**. Pokud je běh programu přerušen uživatelem, přejde stav na **Stopped**. Stav **Failed** označuje, že při spuštění nebo běhu nastala chyba, kterou je možné dohledat v logu. Aktuální stav programu (včetně zachyceného logu) je zobrazen v uživatelském rozhraní v rámci konfigurační karty systému (viz kapitola 6.8). Důvody stavu **Failed** mohou být například:

- obsazení požadovaného síťového portu jiným programem,
- nesprávná verze,
- chybné označení kamery/sériového portu,
- nedostupná kamera/sériový port (využití jiným programem),
- nedostatečná práva pro přístup ke kameře/sériovému portu,
- neplatné rozlišení pro kameru,
- odpojení potřebného hardware za běhu.

Samotný program je z pohledu zdrojového kódu rozdělen do několika částí, které odpovídají názvům souborů se zdrojovými kódy projektu. Různé moduly jsou částečně závislé na platformě a jsou rozděleny pomocí řídících direktiv preprocesoru.

camera Modul umožňuje zachytávat snímky ze zvolené kamery a zasílat je skrze spojení klientovi. Zaznamenaný obraz je v barevném tříkanálovém formátu (`R8G8B8`), volitelně je možné získat i rozdíl aktuálního a předchozího snímku. V takovémto případě je výsledný formát obrazu pouze jednokanálový (`R8`). Rozlišení všech snímků je stejné a lze ho nastavit při inicializace tohoto modulu, výchozím rozlišením je 960×1280 s ohledem na výchozí hardware. Vzhledem k umístění kamery v konstrukci (na výšku) je obraz automaticky vrácen přetočený tak, aby byl vodorovně.

comport Tento modul zprostředkovává zasílání a přijímání dat přes sériovou linku.

horus Modul zpracovává vyšší funkcionalitu pro komunikaci s firmware původního skeneru Ciclop. Poskytuje funkce, které zasílají příslušné `GCode` pro ovládání laserů a krokového motoru, skrze sériovou linku.

log Poskytuje funkce pro logování zpráv. Funkce `log` přijímá pouze řetězec, funkce `logf` se z pohledu formátování chová obdobně jako `printf`.

main Vstupní bod programu. Prvním krokem je spuštění síťového serveru pro navázání komunikace s klientem. Navázání komunikace má přesná pravidla, popsaná v kapitole 6.4. V hlavní smyčce jsou pak očekávány příkazy od klienta.

rpigpio Modul poskytuje funkce pro řízení hardwaru (laserů a krokového motoru) skrze GPIO port Raspberry Pi 3.

server Slouží pro multiplatformní přístup k síťovému rozhraní. Umožňuje zachytit socket naslouchánim na zvoleném portu a řídit příjem a zápis dat.

6.4 Komunikace

Pro komunikaci mezi výše popsanými projekty byl navržen jednoduchý komunikační protokol. Komunikace probíhá přes navázané TCP spojení (viz kapitola **??**) pomocí krátkých zpráv. Komunikaci zahajuje P.S.T.R.O.S. (v diagramu označen jako `Go`), přičemž P.S.T.R.O.S.i.C.E. (v diagramu označena jako `C`), naslouchá na dané adrese a portu, a čeká na připojení. Tyto zprávy mají předem definovanou podobu a jejich zasílání je uskutečňováno v přesně daném pořadí. Celý průběh zahájení komunikace je vidět na obrázku 9. Zde jsou vyobrazeny všechny možné případy, které mohou během komunikace nastat, a jak je na ně reagováno.

Pro jednoduchost je celá komunikace realizována pomocí několika málo základních prvků, konkrétně: *zprávy*, *čísla*, *páry*, *řetězce* a *data*. Zpráva je vždy 32bitová a jejich výpis (včetně jejich významu a datové reprezentace) je uveden v tabulce 6. Sloupec data obsahuje samotnou

Tabulka 6: Výpis komunikačních zpráv

Zpráva	**Parametr**	**Funkce**	**Data**
Hi	ne	Inicializační zpráva.	"`Hi!`"
Bye	ne	Ukončení komunikace.	"`Bye`"
OK	ne	Potvrzovací zpráva.	"`OK!`"
Bad	ne	Negativní zpráva.	"`Bad`"
Snap	ne	Pořízení snímku z kamery.	"`[O]`"
SnapDiff	ne	Pořízení odečteného snímku.	"`[X]`"
Clip	2x *pár*	Nastavení oříznutí snímků.	"`[#]`"
LaserOn	ne	Zapnutí laseru.	"`<--`"
LaserOff	ne	Vypnutí laseru.	"`<  `"
MoveMotor	*číslo*	Pohyb motoru.	"`(@)`"

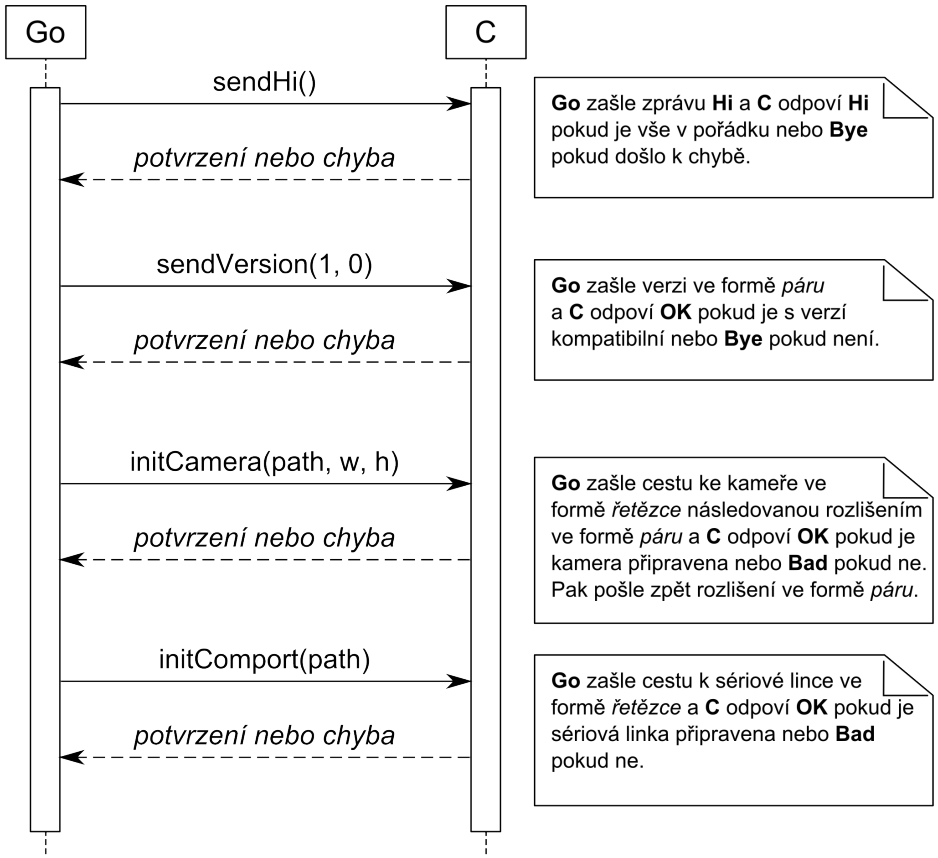
zprávu ve formě řetězce zakončeného nulou, přičemž bajty jsou v pořadí *little endian*, které platí pro celou komunikaci. *Číslo* je vždy 32bitové a bez znaménka. Datový typ *pár* je složen ze dvou 16bitových čísel bez znaménka, nejčastěji značených jako X a Y. Jsou přenášeny vždy dohromady ve formě 32bitového čísla, a to v opačném pořadí (Y a pak X) s ohledem na endianitu. *Řetězce* jsou přenášeny v kódování UTF-8, předcházeny jedním bajtem, obsahujícím délku následujícího řetězce. Řetězec je vždy zakončen znakem s kódem nula, který se nezapočítává do zasílané délky. Maximální délka samotného řetězce je tedy 255 znaků, nejkratší řetězec má pak délku 0 (reprezentován dvěma bajty s nulovou hodnotou přímo za sebou). *Data* pak představují blok blíže nespecifikovaných bajtů přímo za sebou.

Jakmile je zahájení komunikace zcela dokončeno, program očekává další zprávu. Pro zprávy *Snap* a *SnapDiff* je v opačném směru zaslán požadovaný obrázek, ve tvaru:

- *zpráva* "`cam`",
- *číslo* určující počet bajtů obrázku,
- *data* obrázku,
- *zpráva* "`end`".

Zpráva *Clip* je následována dvěma *páry* představující levý horní roh ořezu a jeho výšku a šířku (v tomto pořadí). Pokud jsou dosazeny čtyři nuly, ořezávání se vypne a poté je opět zasílán obraz v plném rozlišení. Stav oříznutí se uchovává po celou dobu komunikace až do dalšího volání *Clip* nebo případného restartu programu. Ve výchozím stavu je obraz zasílán neořezávaný. Pokud bylo ořezávání přijato, je volání zprávy *Clip* potvrzeno zprávou *OK*. V jiném případě je zpět zaslána zpráva *Bye*.

Zpráva *MotorMove* je následována číslem, představujícím úhel pootočení motoru. Není-li řečeno jinak, není tato ani další zpráva potvrzována zpětným zasláním jakékoli zprávy.



Obrázek 9: Průběh komunikace mezi projekty

6.5 Konfigurace

Pro individuální nastavení systému je možné měnit některé jeho parametry. To je uskutečňováno přes konfigurační nabídku v klientovi (viz kapitola 6.8). Jedná se konkrétně o možnost:

- nastavení cesty ke kameře (například `/dev/video0`),
- nastavení rozlišení kamery,
- nastavení cesty k sériovému portu (například `/dev/ttyUSB0`),
- volby síťového portu (například 8888),
- nastavení pokusů o připojení,
- nastavení doby spojení (v milisekundách),
- volby výchozích kalibračních hodnot.

Všechny tyto hodnoty jsou ukládány do souborů ve formátu `JSON`. Jedná se o odlehčený textový, na jazyce zcela nezávislý formát pro výměnu dat, jehož výhodou je jednoduchá zapisovatelnost a následná analýza. Většina programovacích jazyků, včetně námi využívaného jazyka

`Go`, má již zabudované generátory a parsery tohoto formátu. Všechny zvolené konfigurace jsou ukládány do domovské složky uživatele, přičemž při každé změně je automaticky provádí záloha předešlé verze pro případ chyby v ukládání.

Při prvotním spuštění systému jsou příslušné volitelné hodnoty vyzvednuty z výchozího konfiguračního souboru. Ten je v případě spouštění na operačním systému Windows skrytý, v případě operačního systému Linux se tento soubor nachází mimo adresář projektu.

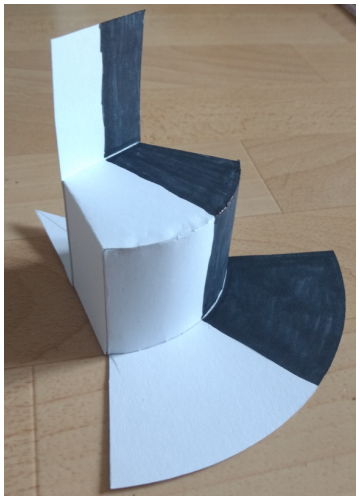
Na závěr poznamenejme, že v případě nulového (tedy uložení prázdného řetězce) nastavení cesty k sériovému portu je využíván modul `rpigpio`, popsaný v kapitole 6.3. Jednotlivé periferie skeneru se tedy očekávají na GPIO portu řídící jednotky Raspberry Pi.

6.6 Kalibrace

Výsledná kalibrace má v našem případě šest hodnot. Při rekonstrukci 3D tvaru objektu je v rámci této práce využíván algoritmus popsaný v kapitole 6.7. V něm je využíváno válce jako projekčního tělesa, kdy promítaná linka laseru vyznačuje část roviny (polygon), ve které je hledán tvar objektu (viz obrázek 11). Kalibrační hodnoty představují právě koordináty tohoto polygonu. Jedná se o souřadnici levého a pravého horního vrcholu polygonu, levého a pravého dolního vrcholu polygonu a vertikální vzdálenosti mezi nimi. Poznamenejme, že souřadnice vrcholů jsou brány od levého horního bodu obrazu.

6.6.1 Manuální kalibrace

V rámci této práce je uživateli poskytnuta manuální kalibrace, při které má možnost ručně zvolit daný polygon na konkrétním (aktuálním) snímku získaného z kamery. Kalibrační hodnoty jsou vypočítány automaticky a vždy je zachována rovnoběžnost (válec). Pokud uživatelem není zvolena žádná konkrétní kalibrace, systém využívá výchozí kalibraci, kterou je možné změnit v klientské konfigurační kartě, popsané v kapitole 6.8.



Obrázek 10: Předmět pro automatickou kalibraci

6.6.2 Automatická kalibrace

Kromě manuální kalibrace poskytuje výsledné řešení i možnost automatické kalibrace. Ta probíhá za pomoci kalibračního předmětu, který je potřeba umístit na skenovací plošinu. Kalibrační předmět je zobrazen na obrázku 10. Jelikož skener zná velikost a proporce jednotlivých částí kalibračního předmětu, je schopen na základě jeho obrazu určit kalibrační hodnoty.

Kalibrační předmět je možné jednoduše vyrobit ze čtvrtky tvrdého papíru za pomoci přiložené šablony. Šablona je součástí digitální přílohy této práce ve formátu PDF.

6.7 Rekonstrukce objektu

Další částí této práce je rekonstrukce 3D tvaru objektu ze získaných snímků. V kapitole jsou popsány principy získávání mračna bodů, způsob jeho ukládání a následné vizualizace uživateli.

6.7.1 Proces skenování

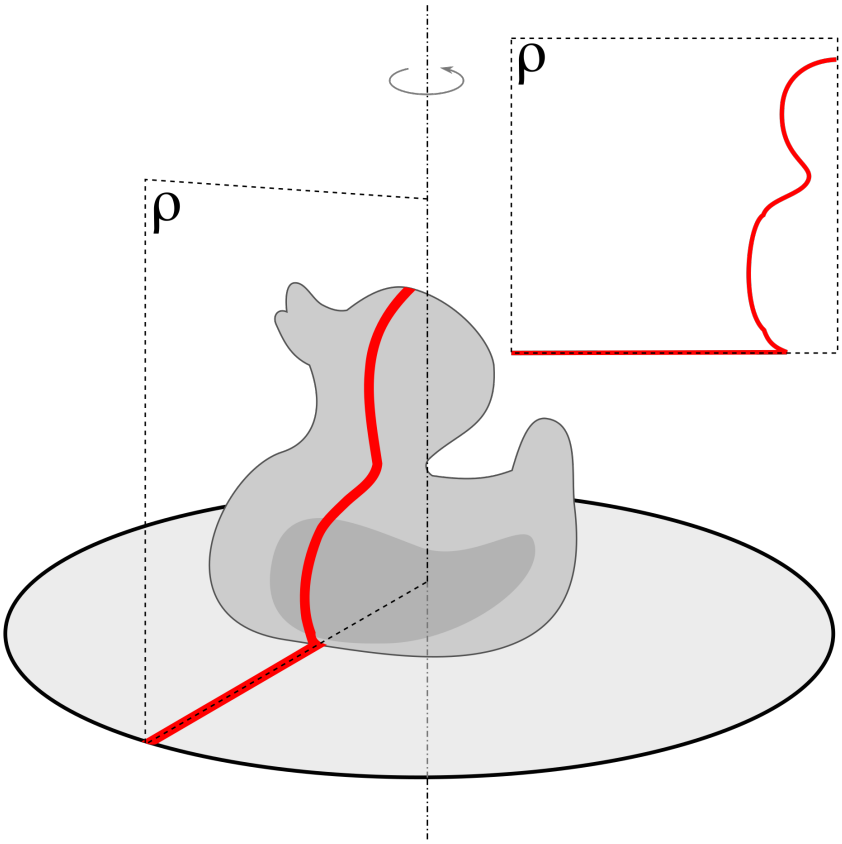
Skenování je řízeno balíčkem `scanning` v projektu P.S.T.R.O.S. Zdrojem snímků je v tomto případě libovolný objekt, splňující rozhraní `scanning.Scanner`. Algoritmus pro získání bodů je opět řešen obecně pomocí funkce, odpovídající datovému typu `scanning.Algorithm`. Skenovací proces se vždy sestává z následující posloupnosti kroků, které se opakují pro každý řez objektem:

1. získání snímku z kamery,
2. zapnutí laseru,
3. získání rozdílového snímku z kamery,
4. vypnutí laseru,
5. zpracování snímků rozpoznávajícím algoritmem,
6. pootočení objektu za pomoci motoru.

Poznamenejme také, že laser je na začátku skenovacího procesu vypnutý. Jakmile je skenovací proces ukončen (ať už došlo k úspěšnému dokončení skenování či nikoli), uloží se získané mračno bodů do úložiště.

6.7.2 Rekonstrukce tvaru objektu

Výsledné mračno bodů, představující 3D rekonstrukci skenovaného objektu, vznikne složením jednotlivých řezů, kdy každý řez odpovídá dvojici snímků (původní snímek a snímek s odečtením laseru od okolí). V rámci těchto snímků je potřeba identifikovat linii laseru na objektu a převést její body z 2D prostoru snímku do 3D. Způsob, jakým body vzniklé řezem laseru korespondují, je viditelný na obrázku 11.



Obrázek 11: Schéma fungování skeneru při rekonstrukci 3D tvaru objektu

Tento výpočet zajišťuje samostatná rozpoznávací funkce, která zpracovává každý řez odděleně. Na vstupu se nachází oba získané snímky, úhel otočení objektu (vůči svislé ose otáčení) a kalibrační body (viz 6.6). Výstupem je část mračna bodů, která byla získána a odpovídá jednomu řezu.

Během vývoje výsledného řešení byla rozpoznávací funkce několikrát přepracována a optimalizována. Ve výsledné verzi softwaru je k dispozici několik nezávislých rozpoznávacích funkcí (jejich volba není v tuto chvíli možná ze strany uživatele, ale pouze ze strany kódu).

Původní a v tuto chvíli výchozí rozpoznávací funkce nejdříve provede předzpracování rozdílového snímku. To zahrnuje prahování následované operacemi dilatace a eroze obrazu, které mají za úkol odstranit případný šum, vzniklý v důsledku nečistoty obrazu. Pokračuje se operací ztenčení, která má za úkol zpřesnit linku laseru v obraze. Finálním krokem je proložení řezu sadou vodorovných rovin, kdy průsečíkem obrazu laseru a této roviny vznikne požadovaný bod. Ten je přepočten do 3D souřadnic a přidán do mračna bodů.

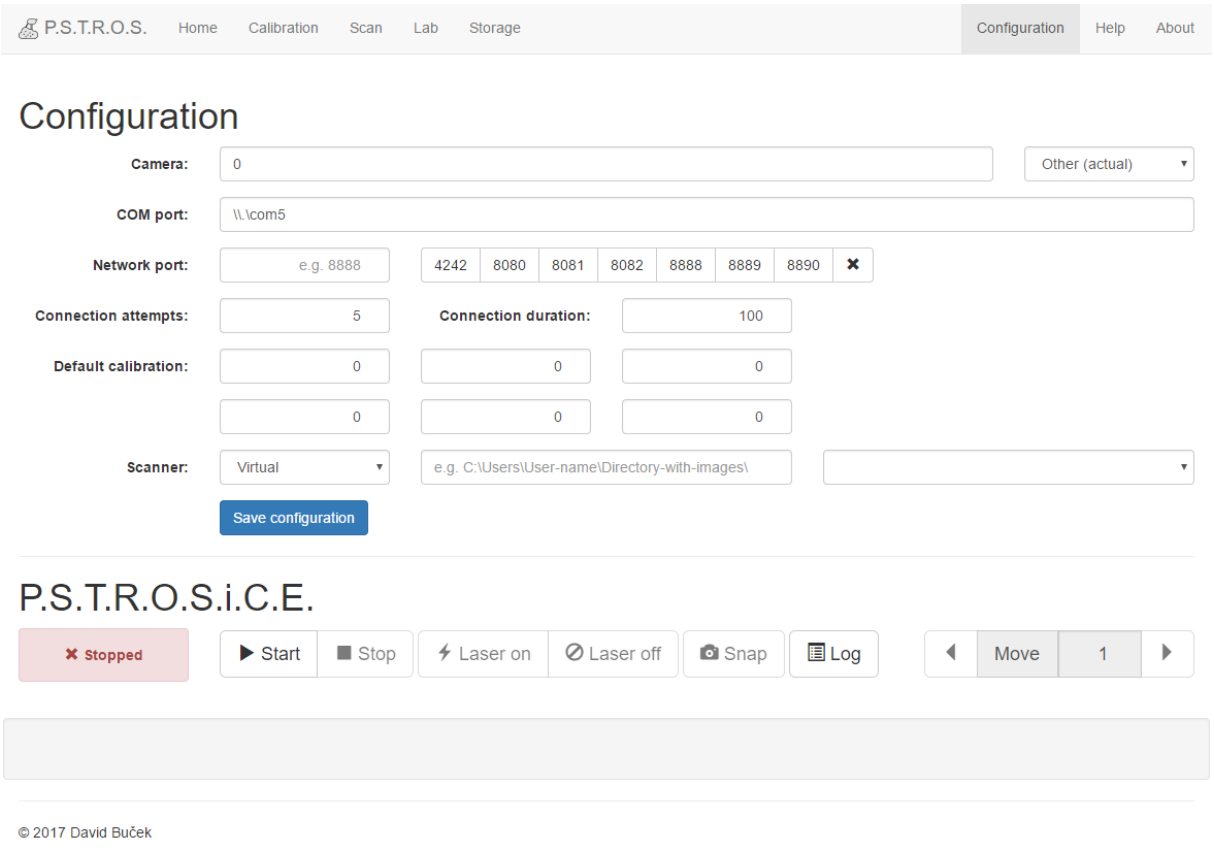
6.7.3 Vizualizace mračna

Pro potřeby zobrazení mračna na straně klienta byla vyvinuta vlastní `HTML5` komponenta. Tu je možné integrovat do jakékoliv webové stránky a je celá umístěna v jednom souboru.

Ke zobrazení bodů využívá technologii `WebGL` ve verzi 1. Mračno je možné otáčet a přibližovat pomocí myši. Data je možné do komponenty přidávat průběžně. Komponenta pracuje s formátem PPP ve variantě Base64, popsaným v podkapitole 4.4.3.

6.8 Uživatelské rozhraní

Celé uživatelské rozhraní skenovacího systému je rozděleno do osmi nabídek. Jedná se o domovskou, kalibrační, skenovací, editační, úložišťovou a konfigurační nabídku. Ty jsou dále doplněny o nabídku nápovědy a informací o celém systému.



Obrázek 12: Ukázka grafického uživatelského rozhraní systému

Úvodní domovská nabídka s označením **Home** je otevírána při spuštění systému. Její účel je čistě informativní, stránka obsahuje název systému a vizualizer mračna bodů.

Kalibrační nabídka (označení **Calibration**) slouží k volbě a editaci kalibračních hodnot (viz kapitola 6.6). Celkem se jedná o šest položek, které je možné navolit z vyobrazených textových polí. Takovouto kalibraci je možné následně uložit anebo naopak vyresetovat kalibrační hodnoty

do původní (výchozí) podoby. Kalibrace je vždy brána vůči aktuálnímu rozpoložení objektu, proto je možné z této nabídky zachytit aktuální snímek z kamery a kalibraci poté nakonfigurovat vůči němu. Kromě zadání všech šesti kalibračních čísel je rovněž možné provést automatickou kalibraci, která je popsaná v kapitole 6.6.2).

Zřejmě nejpodstatnější nabídkou je karta **Scan** sloužící pro zahájení skenovacího procesu. Uživatel je jako první vyzván ke zvolení názvu výsledného mračna (instrukce k volbě názvu jsou vytyčeny v rámci poznámek na této nabídce), a teprve po jeho zvolení je možné proces spustit volbou *Star scanning*. Tato nabídka nabízí náhled v reálném čase na dosavadní průběh skenování. Postupně tvořené mračno je vizualizováno v pravé části této stránky. Tuto možnost je však možné vypnout z důvodu vyšších výpočetních nároků. Nezávisle na této volbě je uživateli zobrazován aktuální počet vytvořených bodů mračna. Skenovací proces lze kdykoli ukončit, a to pomocí volby *Stop scanning*.

Další nabídka, s označením **Lab**, slouží k editaci a vylepšení již vyprodukovaného mračna. Její aktuální účel je však pouze k vizualizaci jednoho z mračen umístěných v úložišti systému. Rozšíření této nabídky je součást dalších vylepšení tohoto systému (viz kapitola 7.3).

Úložiště (nabídka **Storage**) obsahuje všechny mračna, která jsou součástí systému. Jednotlivá mračna jsou vypsána v tabulce, kdy ke každému mračnu je zobrazena informace o datu jeho vytvoření a počtu bodů v něm. Každé mračno je dále doplněno o nabídku akcí, zahrnující jeho vizualizaci (v rámci nabídky *Lab*), možnost jeho exportu do jednoho ze tří podporovaných formátů (`PPP`, `OBJ` a `PLY`) a nakonec jeho permanentní smazání z úložiště.

Velmi důležitou funkci plní nabídka konfigurace (označení **Configuration**). V ní je uživateli umožněno nastavení celého systému včetně výběru skeneru (hardwarový nebo virtuální, viz 6.2.7). Další možná volitelná nastavení jsou popsána výše jako součást kapitoly 6.5.

V rámci této nabídky je rovněž prováděno spouštění projektu P.S.T.R.O.S.i.C.E (kapitola 6.3). Po jeho úspěšném startu a navázání komunikace je uživateli umožněno systém otestovat. Je zde možnost zapnutí a vypnutí laserů, pořízení snímku z kamery a pohybu krokového motoru. Veškeré prováděné akce, a také i vzniklé chyby, jsou zachytávány do logovacího souboru, který je možné zobrazit volbou *Log* v této nabídce.

Zbývající dvě karty **Help** a **About** není nutné podrobněji popisovat, jelikož jejich smysl je čistě informativní a není skrze ně možné provádět jakékoli interakce se systémem.

6.9 Instalace a spuštění systému

Celý systém byl navržen tak, aby byl co nejvíce univerzální a multiplatformní. Způsoby nasazení na jednotlivé operační systémy a platformy jsou popsány v následujících podkapitolách.

Bez ohledu na způsob spuštění projektu P.S.T.R.O.S. je možné mu předat spouštěcí parametry. Příklad předání parametrů: `pstros -log /var/log -conf /etc/pstros`.

Parametr `log` obsahuje cestu ke složce, kde se má vytvořit logovací soubor (`pstros.log`). Pokud není parametr přítomen, použije se aktuální složka, ve které se program nachází. Soubor

je otevřen pro zápis a pokud již existuje, je přidáváno na jeho konec. Pokud se soubor nepodaří založit, probíhá výpis logů na standardní výstup programu.

Parametr `conf` obsahuje cestu ke složce s konfiguračním souborem (`pstros.conf`). Stejně jako u předchozího parametru způsobí jeho nepřítomnost použití aktuální složky. Soubor se otevírá jen v době načítání nebo ukládání konfigurace. Konfigurační soubor je formátu JSON a v základu obsahuje objekt. Klíčovou vlastností je položka `pstros_path`, obsahující cestu ke spustitelnému souboru projektu P.S.T.R.O.S.i.C.E.

6.9.1 Spuštění na Windows

Pro kompilaci celého softwaru ze zdrojových kódů na operačním systému Microsoft Windows (testováno na verzi 10) je potřeba následující:

1. vývojová sada jazyka `Go` ve verzi alespoň 1.8,
2. překladač jazyků `C` a `C++` (Visual Studio 2013),
3. knihovna `OpenCV` ve verzi 3.2.0,
4. zdrojové kódy projektů P.S.T.R.O.S. a P.S.T.R.O.S.i.C.E.

Prvním krokem je nainstalovat vývojovou sadu jazyka `Go`, kterou je možné stáhnout na adrese `https://golang.org/dl/`. K dispozici je instalační program, který zavede všechny potřebné nástroje (ty jsou pak dostupné skrze příkazovou řádku). Překladač `Go` navíc od verze 1.8 automaticky očekává `GOPATH` na adrese `%USERPROFILE%\go` bez nutnosti ho zavádět do proměnného prostředí systému [19]. `GOPATH` je pracovní složka `Go`, obsahující tři základní podsložky:

- `bin` pro výsledné binární soubory,
- `pkg` pro jednotlivé balíčky jazyka,
- `src` pro zdrojové kódy včetně vlastních projektů.

Do složky `src` přijde samotný projekt P.S.T.R.O.S., výsledná cesta k souboru `main.go` může tedy vypadat následovně: `C:\Users\David\go\src\main.go`. Pro spuštění stačí zapnout `start.bat` umístěný v kořenové složce projektu. Jelikož výsledná aplikace využívá port 80 k poskytování grafického uživatelského rozhraní, je potřeba příslušně nastavit firewall a další omezující software.

Pro sestavení projektu P.S.T.R.O.S.i.C.E. lze využít Visual Studio 2013 a vyšší. Projekt obsahuje příslušný `sln` soubor, který stačí spustit a sestavit. Podmínkou je nastavení některých systémových proměnných. Proměnná `GOPATH` musí vést na adresu s pracovní složkou `Go` (pro sestavení tohoto projektu je nutné ji explicitně nastavit vždy). Dále musí být v projektu nainstalováno `OpenCV` ve verzi 3.2.0 nebo kompatibilní a systémová proměnná `OpenCV320` musí vést do složky `build`. Bitová verze knihovny musí být shodná s bitovou verzí programu a příslušné `dll` soubory se musí nacházet tak, aby je program při spuštění nalezl.

6.9.2 Spuštění na Linux

Sestavení pro operační systém Linux probíhá obdobně jako pro již zmíněný systém Microsoft Windows. Postup je uzpůsoben pro distribuci `Debian` nebo z ní odvozené, lze však upravit i pro jiné distribuce. Nutné součásti pro spuštění jsou:

1. vývojová sada jazyka `Go` ve verzi alespoň 1.8,
2. překladač jazyka `C` (`gcc` a `make`),
3. knihovna `v4l2` (včetně `dev` souborů),
4. zdrojové kódy projektů P.S.T.R.O.S. a P.S.T.R.O.S.i.C.E.

Instalace a nastavení `Go` je popsáno v podkapitole 6.9.1 a pro Linux je obdobné. Automaticky ji lze provést i pomocí balíčkovacího systému příkazem `sudo apt-get install golang-1.8-go` nebo pomocí postupu na adrese `https://github.com/golang/go/wiki/Ubuntu`. Pro spuštění stačí zapnout `start.sh` umístěný v kořenové složce projektu. Opět je potřeba zohlednit firewall a další.

Sestavení projektu P.S.T.R.O.S.i.C.E. je možné pomocí utility `make`. Ke správnému sestavení je nutné mít zavedenou systémovou proměnnou `GOPATH` a nainstalovanou knihovnu `v4l2`. Tu je možné získat pomocí příkazu: `sudo apt-get install libv4l-dev`.

Pro přístup k zařízení sériové linky a webové kamery je potřeba správně nastavit přístupová práva. Příklad takovéhoto nastavení může být: `sudo chmod 666 /dev/ttyUSB0`.

6.9.3 Spuštění na Raspberry Pi

Jednou z možností je spustit celý software na Raspberry Pi, které bude sloužit jako *řídící jednotka* celého skeneru (viz 6.1). Aby bylo celé řešení plně autonomní, lze nainstalovat projekt P.S.T.R.O.S. jako službu, která se bude spouštět spolu se systémem. K tomu je potřeba vytvořit dva spustitelné soubory kompatibilní s Raspberry Pi: `pstros` a `pstros-updater`. Ty je možné sestavit na jiném počítači za použití postupu z podkapitoly 6.9.1 nebo 6.9.2 za pomocí dávek `build` ve složkách `/pi` a `/pi/updater`. Přesný postup nasazení služeb je popsán v souboru `/pi/How to configure pstros service.odt`.

Pro nasazení systému nebo jeho nové verze není potřeba na Raspberry Pi instalovat překladač `Go`. Aktualizaci je možné opět provést ze vzdáleného počítači pomocí dávky `/pi/deploy`, která jako první parametr přijímá IP adresu Raspberry Pi.

Projekt P.S.T.R.O.S.i.C.E. je nutné sestavit zvlášť. První možností je sestavit jej přímo na Raspberry Pi podle návodu totožného s postupem v podkapitole 6.9.2. Projekt lze případně sestavit na Linuxu pro jinou platformu, pokud jsou doinstalovány příslušné nástroje.

7 Výsledky a testování

Po implementaci vlastního navrženého řešení je nezbytné výsledný produkt náležitě otestovat. Součástí kapitoly jsou i možné další návrhy na vylepšení stávajícího řešení.

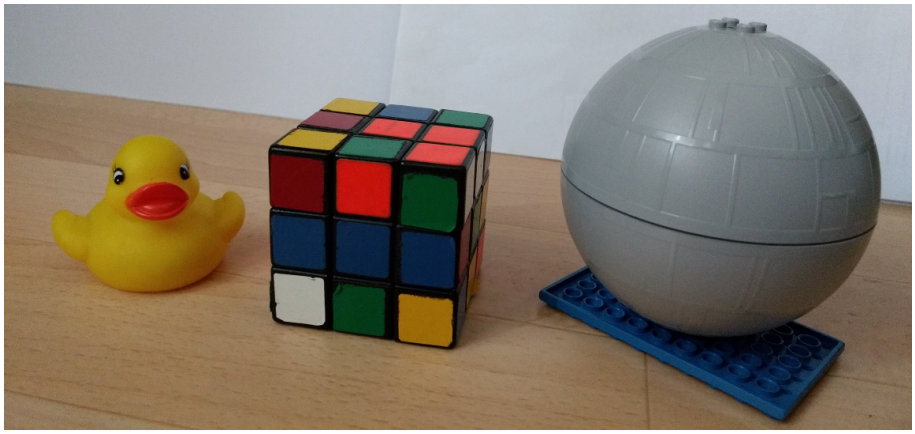
7.1 Testování v průběhu vývoje

V průběhu vývoje byly jednotlivé části softwaru průběžně testovány. Ve většině případů se jednalo o manuální testy dílčích částí systému. Velká důraz byl kladen na testování projektu P.S.T.R.O.S.i.C.E., který je implementovaný v jazyce `C`. Ten slouží jako most k nízkým API operačního systému, proto bylo nutné, aby fungoval zcela bezchybně.

Velká část práce pak probíhala v jazyce `Go`, který už v základu nabízí funkce pro testování softwaru. Obsahuje například podporu pro automatické testovaní, která je obsažena v balíčku `testing`. Testy je pak možné provádět přímo z příkazové řádky pomocí nástroje `go test`. Tento nástroj poskytuje i podporu pro pokrytí kódu, kterou je možné zapnout přepínačem `go test -cover`.

7.2 Testování výsledného řešení

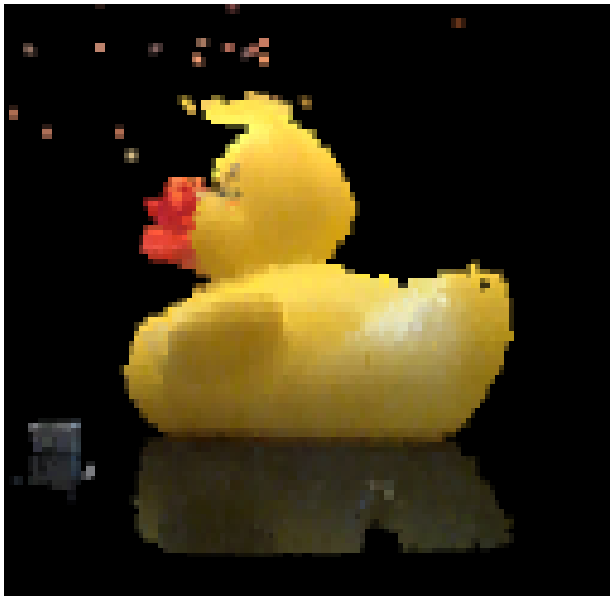
Při testování výsledného řešení byly zvoleny čtyři referenční objekty, které byly naskenovány. Jejich volba probíhala s ohledem na to, aby bylo ověření skeneru co nejkomplexnější. Za referenční objekty byly zvoleny předměty z obrázků 13 a 18: kachnička, kostka, koule a přebal alba s textem.



Obrázek 13: Referenční objekty

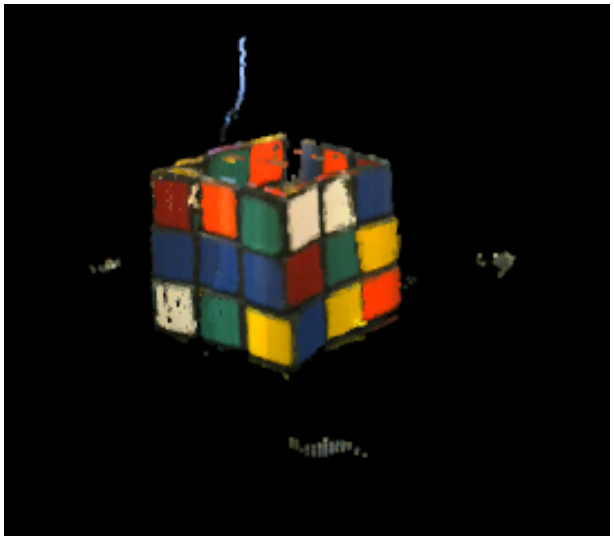
Předměty, které je skener schopen zpracovat, se musí vejít na rotační plošinu a zorného pole kamery. Průměr plošiny je 20 cm a s ohledem na pozici kamery a laseru je maximální výška předmětu okolo 15 cm. Omezení se nevztahují pouze na rozměry ale i na hmotnost objektu. Jelikož rotační plošinou otáčí krokový motor, neměla by hmotnost objektu přesáhnout únosnou mez.

První pokusy, vztahující se k počátkům vývoje, byly prováděny s referenčním objektem kachničky. Volba vychází ze tvaru objektu, který je podobný kouli, ale má dostatek nepravidelností (křídla, hlava, zobák), více barev a jeho povrch je matný. Sken tohoto referenčního objektu je zaznamenán na obrázku 14.



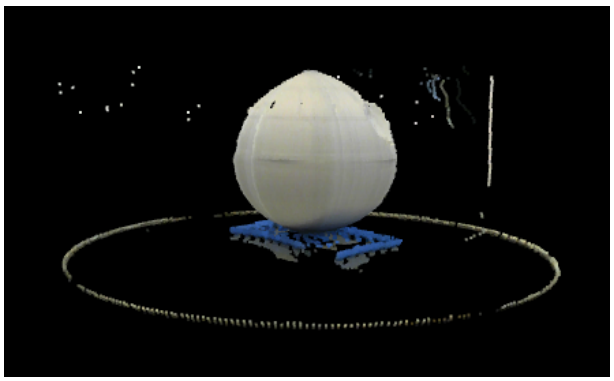
Obrázek 14: Naskenovaný referenční objekt - kachnička

V souvislosti se způsoben, kterým skener pracuje, není schopen snímat plochy rovnoběžné s rotační plošinou nebo dutiny. Toto je jasně viditelné na referenčním objektu kostky, jejíž sken je vidět na obrázku 15. Naopak zachycení rekonstrukce referenčního objektu koule nebyl pro skener žádný problém.



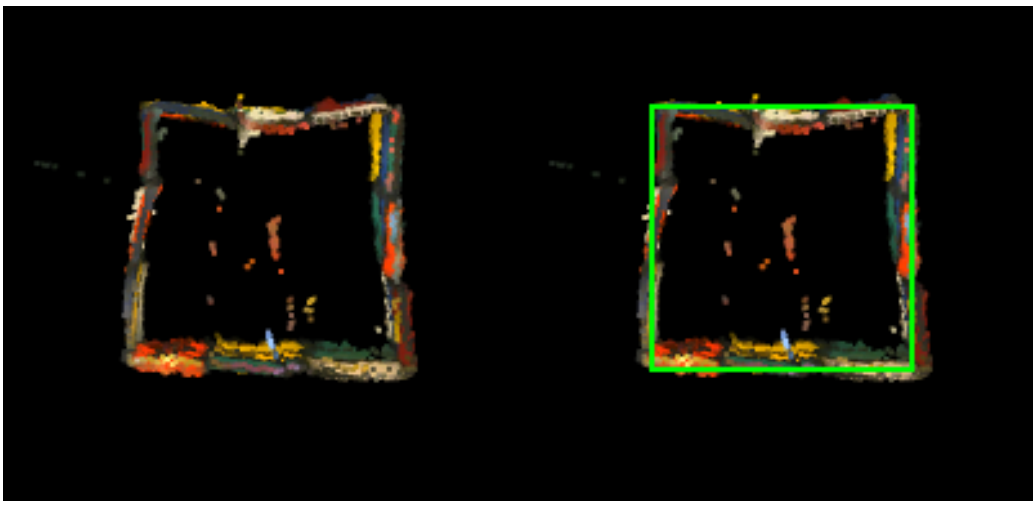
Obrázek 15: Naskenovaný referenční objekt - kostka

Během testů se rovněž zjistilo, že významnou roli hrají okolní světelné podmínky a materiál skenovaného objektu. Kvalita výsledného skenu je závislá na množství okolního světla, přičemž oba extrémy negativně ovlivňují výsledek rekonstrukce. Při příliš slabém osvětlení (šeru) se ztrácí přesnost barvy, které se rekonstruují z prvního snímku (bez použití laseru). Naopak při příliš silném osvětlení, například při testech s umělým přímým světlem, se ztrácí světlo laseru a není možné identifikovat referenční linku.



Obrázek 16: Naskenovaný referenční objekt - koule

Pokud je materiál skenovaného objektu příliš reflexní nebo naopak matný, dojde při osvětlení laserem k poškození referenční linky. Současně i odraz paprsku laseru může způsobit problémy při odečtu obou snímků. Naopak, pokud materiál světlo laseru pohltí, není linka na rozdílovém snímku vidět a skener tedy nezaznamená v tomto místě žádné body.



Obrázek 17: Naskenovaný referenční objekt - kostka, pohled shora

Další měřená vlastnost výsledného mračna byla přesnost zachování původního tvaru objektu. Toto je dobře viditelné na referenčním objektu kostky, jak zachycuje obrázek 17. Na kulatých plochách dalších referenčních objektů nemusí být tento problém dostatečně výrazný. Nepřesnosti mohou vzniknout hlavně nepřesným otáčením rotační plošiny.

Obrázek 18: Naskenovaný referenční objekt - text

Posledním zkoumaným aspektem byla kvalita snímání barev povrchu skenovaného objektu. Z tohoto důvodu byl za referenční objekt přijat přebal alba, který obsahuje jasně čitelný text, který ale není příliš kontrastní. Jak je vidět z obrázku 18, skener si se zachycením textury poradil bez větších obtíží a text je čitelný.

7.3 Další vylepšení

Přestože výsledný software a celkové řešení splňuje všechny požadavky, stále je zde prostor k dalšímu rozvoji projektu nad rámec jeho zadání. Již během vývoje byly zaznamenány možnosti, jak projekt dále vylepšit, a to jak na straně softwaru, tak na straně hardwaru. Následující podkapitoly shrnují případné možnosti vývoje projektu do budoucna.

7.3.1 Konstrukce skeneru

Ačkoli vyvinutý software dokáže pracovat univerzálně a je možné ho rozšířit pro práci s libovolným kompatibilní hardwarem, v rámci této práce byl využit jako základ skener BQ Ciclop. Největší slabinou této konstrukce se ukázal být úchyt motoru k rotační plošině se skenovaným objektem. Ideální by bylo nahradit existující součástku jiným řešením úchytu.

Pro potřeby výsledného řešení je používán pouze jeden z laserů, stejně tak ovládání pomocí Arduina s CNC shieldem je v některých ohledech zbytečně složité. Protože software je možné nainstalovat na mikropočítač Raspberry Pi, který sám disponuje vstupně/výstupními piny, bylo by možné řízení přenechat na něm a Arduino z konstrukce úplně vyřadit. Nabízí se rovněž využití Raspberry Pi kamery místo použité webové kamery.

Možným řešením je pak návrh vlastní konstrukce, který by splňovala všechny požadavky pro použití s výsledným softwarem.

7.3.2 Pokročilá práce s mračny

Součástí řešení je obrazovka `Lab`, která umožňuje zobrazit libovolné mračno z úložiště. Její funkce by se daly rozšířit o možnosti pokročilé práce s mračny, jako jsou základní transformace, skládání mračen nebo filtrování bodů. V takovém případě by bylo vhodné, aby bylo možné mračna do úložiště nahrávat ze vzdáleného počítače.

Kromě úprav by bylo možné přidat možnost triangulace povrchu výsledného mračna a exportovat celé povrchy jako shluky trojúhelníků.

7.3.3 Víceuživatelský přístup

Díky návrhu grafického uživatelského rozhraní, které je realizováno pomocí webového serveru, je možná práce se softwarem ze vzdáleného počítače. V případě rozšíření funkcionality a zajištění synchronizace všech funkcí je možné, aby se systémem pracovalo více uživatelů současně.

8 Závěr

Výsledkem této práce funkční softwarové řešení pro rekonstrukci 3D tvaru objektu. Řešení je multiplatformní, samostatné a navržené s ohledem na další možné rozšiřování.

Teoretická část práce rozebírá existující řešení, která byla otestována v případech, kde to bylo možné. Shrnuje také možnosti, jak vyvinout nový obdobný systém. Jsou zde shrnuty možné varianty a použitelné části.

Výsledný software je napsán v jazyce `Go` a je navržen pro snadnou udržitelnost vývoje a přenositelnost mezi platformami. Odstínění hardwaru je řešeno pomocí jednoduché aplikace, která je napsána v jazyce `C`. Grafické uživatelské rozhraní je vytvořeno formou webu, díky čemuž je dostupné i ze vzdáleného počítače. Software umožňuje pracovat s různými druhy skenerů a lze jej rozšířit o nové. Rekonstruované objekty je možné přímo na webové stránce zobrazit a prohlížet, popřípadě je stáhnout jako mračno bodů v některém z běžně používaných formátů.

Literatura

[1] SOJKA, Eduard. *Digitální zpracování a analýza obrazů* [online]. Ostrava, 2000 [cit. 2017-04-19]. Dostupné z: http://mrl.cs.vsb.cz/people/sojka/dzo/digitalni_zpracovani_obrazu.pdf. Skriptum. Vysoká škola báňská – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky.

[2] SOJKA, Eduard, Martin NĚMEC a Tomáš FABIÁN. *Matematické základy počítačové grafiky* [online]. Ostrava, 2011 [cit. 2017-04-19]. Dostupné z: http://mrl.cs.vsb.cz//people/sojka/pg/pocitacova_grafikaII.pdf. Skriptum. Vysoká škola báňská – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky.

[3] BUČEK, David. *Editor map pro Android*. Ostrava, 2015. Bakalářská práce. Vysoká škola báňská – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky. Vedoucí práce Mgr.Ing. Michal Krumnikl, Ph.D.

[4] Advanced driver assistant systems (ADAS). *Infineon* [online]. [cit. 2017-04-19]. Dostupné z: http://www.infineon.com/cms/en/applications/automotive/safety/adas/

[5] KOVAČOVSKÝ, Tomáš. *SMISS: Scalable Multifunctional Indoor Scanning System*. Bratislava, 2010. Bakalářská práce. Univerzita Komenského v Bratislavě, Fakulta matematiky, fyziky a informatiky. Vedoucí práce Mgr. Ján Žižka.

[6] Gray code. *Wolfram mathworld* [online]. 2016 [cit. 2017-04-19]. Dostupné z: http://mathworld.wolfram.com/GrayCode.html

[7] Kickstarter. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-04-19]. Dostupné z: https://en.wikipedia.org/wiki/Kickstarter

[8] Structured Light 3D Scanning. *Instructables* [online]. [cit. 2017-04-19]. Dostupné z: http://www.instructables.com/id/Structured-Light-3D-Scanning/

[9] The $30 3D scanner V3 updates. *Thingiverse* [online]. 2016 [cit. 2017-04-19]. Dostupné z: http://www.thingiverse.com/thing:1762299

[10] Presentation: Ciclop and Horus. *Diwo* [online]. 2015 [cit. 2017-04-19]. Dostupné z: http://diwo.bq.com/en/presentacion-ciclop-horus/

[11] Ciclop 3D Scanner. *Thingiverse* [online]. 2015 [cit. 2017-04-19]. Dostupné z: http://www.thingiverse.com/thing:740357

[12] Ciclop: Ciclop 3D Scanner. *RepRap* [online]. 2015 [cit. 2017-04-19]. Dostupné z: http://reprap.org/wiki/Ciclop

[13] C++ Versions. *Cplusplus* [online]. 2017 [cit. 2017-04-19]. Dostupné z: http://www.cplusplus.com/site/versions/

[14] Visual C++ Team Blog: C++11/14/17 Features In VS 2015 RTM. *MSDN* [online]. 2015 [cit. 2017-04-19]. Dostupné z: http://www.cplusplus.com/site/versions/

[15] *The Go Programming Language* [online]. 2017 [cit. 2017-04-19]. Dostupné z: https://golang.org/

[16] W3schools. *HTML5 Introduction: What is New in HTML5?* [online]. [cit. 2017-04-19]. Dostupné z: https://www.w3schools.com/

[17] KRAČMAR, Stanislav a Jiří VOGEL. *Programovací jazyk C: Doplňkové skriptum* [online]. Praha, 1995 [cit. 2017-04-19]. Dostupné z: http://fsinet.fsid.cvut.cz/cz/U201/skrc.html. Skriptum. ČVUT.

[18] *The Go Programming Language: Command cgo* [online]. 2017 [cit. 2017-04-19]. Dostupné z: https://golang.org/cmd/cgo/

[19] SettingGOPATH. *GitHub* [online]. 2017 [cit. 2017-04-19]. Dostupné z: https://github.com/golang/go/wiki/SettingGOPATH

[20] *OpenCV* [online]. 2017 [cit. 2017-04-19]. Dostupné z: http://opencv.org/

[21] Linux TV. *Video for Linux Two API Specification: Revision 2.6.32* [online]. [cit. 2017-04-19]. Dostupné z: https://www.linuxtv.org/

[22] GitHub. *Horus* [online]. [cit. 2017-04-19]. Dostupné z: https://github.com/bqlabs/horus

[23] PLY - Polygon File Format. *Paul Bourke - Personal Pages* [online]. [cit. 2017-04-19]. Dostupné z: http://paulbourke.net/dataformats/ply/

[24] B1. Object Files (.obj). [online]. [cit. 2017-04-19]. Dostupné z: http://www.martinreddy.net/gfx/3d/OBJ.spec

[25] PCL: Point Cloud Library. *Pointclouds* [online]. [cit. 2017-04-19]. Dostupné z: http://pointclouds.org/

[26] CzechDUINO: První český Arduino obchod. *Co je to Arduino* [online]. [cit. 2017-04-19]. Dostupné z: http://czechduino.cz/?co-je-to-arduino,29,

[27] Info Trend. *CNC shield Arduino, zapojení , ladění* [online]. [cit. 2017-04-19]. Dostupné z: http://www.infotrend.cz/home/cnc-shield-arduino-zapojeni-ladeni/

[28] Divo. *ZUM SCAN released under CC-BY-SA license* [online]. [cit. 2017-04-19]. Dostupné z: http://diwo.bq.com/en/zum-scan-released-2/

[29] RPiShop: Váš dodavatel Raspberry Pi. *Raspberry Pi 3 Model B 64-bit 1GB RAM* [online]. [cit. 2017-04-19]. Dostupné z: http://rpishop.cz/kategorie/283-raspberry-pi-3-model-b-64-bit.html

[30] *Bootstrap* [online]. [cit. 2017-04-19]. Dostupné z: http://getbootstrap.com/

[31] Pcread. In: *MathWorks: Documentation* [online]. [cit. 2017-04-19]. Dostupné z: https://www.mathworks.com/help/examples/vision/win64/ReadAPointCloudFromAPLYFileExample_01.png